International Journal of Organizational and Collective Intelligence
Volume 12 • Issue 4

Self-Adaptation Through Reinforcement Learning Using a Feature Model

Selma Ouareth, LICUS Laboratory, 20 Août 1955-Skikda University, Algeria*
https://orcid.org/0000-0002-3595-5339

Soufiane Boulehouache, LICUS Laboratory, 20 Août 1955-Skikda University, Algeria & LIRE Laboratory, Adbelhamid Mehri University, Constantine, Algeria

Mazouzi Smaine, LICUS Laboratory, 20 Août 1955-Skikda University, Algeria
https://orcid.org/0000-0003-3587-7657

ABSTRACT

Typically, self-adaptation is achieved by implementing the MAPE-K Control Loop. The researchers agree that multiple control loops should be assigned if the system is complex and large-scale. The hierarchical control has proven to be a good compromise to achieve SAS goals, as there is always some degree of decentralization but it also retains a degree of centralization. The decentralized entities must be coordinated to ensure the consistency of adaptation processes. The high cost of data transfer between coordinating entities may be an obstacle to achieving system scalability. To resolve this problem, coordination should only define between entities that require communication between them. However, most of the current SAS relies on static MAPE-K. In this article, authors present a new method that allows changing the structure and behavior of loops. Authors base on exploration strategies for online reinforcement learning, using the feature model to define the adaptation space.

KEYWORDS

Component Model, Coordination, Exploration Strategies, Feature Model, Hierarchical Control, MAPE-K Control loops, Reinforcement Learning, Self-Adaptive Software Systems

INTRODUCTION

Self-Adaptive System (SAS) have been subject of considerable research effort in different application domain like smart vehicles and smart home. SAS is a system with the ability to automatically modify its structure and behavior at runtime in order to respond to changes in its environment and the system itself (Barna et al., 2009). What distinguishes SAS is its ability to improve its performance under changing operating conditions and to manage itself when certain components fail. Typically, self-adaptation is achieved an integrated a MAPE-K control loop on top of the Managed sub-system (Dobson et al., 2006). The researchers agree that multiple control loops should be assigned if the

DOI: 10.4018/IJOCI.312226 *Corresponding Author

system is complex and large-scale (Vromant et al., 2011) (Weyns et al., 2013) (Calinescu et al., 2015) (Al-Rahmi et al., 2015) (Sylla et al., 2017) (Hachicha et al., 2017) (Ouareth et al., 2020), (Ouareth et al., 2021). Over the past decade, the SAS design process has undergone significant evolution. Indeed, achieving self-adaptation through multiple control loops remains one of the major challenges in SAS that engineers may face.

In the literature, many researchers (Weyns et al., 2013) (Sylla et al., 2017) (Ouareth et al., 2021) (Zavala et al., 2020) choose to rely on hierarchical control (which relies on centralized and decentralized techniques) to improve the overall reliability of the system. However, managing coordination between decentralized MAPE entities can be a major challenge for engineers, especially the high cost of coordination. Additionally, in cases where coordination is required between the MAPE entities of many loops, the scalability of the system may be compromised. On the other hand, most of the current SAS relies on static MAPE-K (Zavala et al., 2020), where the structure and behavior of MAPE-K entities must be defined at the design stage, i.e. it is impossible to make changes to the loop at runtime.

In this article, the authors contribute to extending an approach HCLs (Hierarchical Control Loops), previously proposed at a conference (Ouareth et al., 2021). The aim of this research is to propose a solution that allows the insertion or deletion of coordination between MAPE entities based on the developments that may occur in the system in order to avoid the problem of conflicting decisions and to reduce the cost of coordination. In this case, control loops become dynamic and can change their structure and thus their behavior in response to changes in the system itself as well as their environment.

Recently, researchers focused on the use of Reinforcement Learning (RL) to support self-adaptation in several studies (Quin et al., 2019) (Metzger et al., 2022) (Weyns et al., 2022). These works focused on reducing the adaptation space of SASs that are subject to uncertainties and ensuring efficient decision-making. In this study, the authors relied on exploration strategies for online RL, using the feature model which represents the MAPE-K entities and the interaction relation between different entities in order to define an adaptation space. Also, the authors choose the Q-Learning algorithm to integrate the exploring strategies in order to determine possible configurations on the autonomic manager.

The remainder of this article is structured as follows. The second section presents the background knowledge that allows understanding the paper. Next, the authors describe a method that makes it possible to adapt the loops based on exploration strategies for online reinforcement learning using the feature model. Then, the article presents an airport wireless access case study. Therefore, the authors present a related work. Finally, this paper concludes with a conclusion summarizing the contribution and perspective that could be developed in the future.

BACKGROUND

In this section, the authors introduce the background knowledge that allows understanding the contribution.

Feature Models

Feature Models (FM) (Arcaini et al., 2015) allow the listing of the features as a hierarchical tree, where each relation between the parent and their children can be Mandatory, Optional, or Alternative (Or, XOr). In addition, there are other extra-constraints that specify the relation between different features such as require and exclude relation. It is possible to represent those relations using the propositional formula where: “p” presents the parent, and “a”, “b” presents the different features.

- **Mandatory**: $(p \rightarrow a) \wedge (a \rightarrow p)$
- **Optional**: $a \rightarrow p$
- **Or**: $p \rightarrow (a1 \vee ... \vee an) \wedge (a1 \rightarrow p) \wedge ... \wedge (an \rightarrow p)$

- XOr: (p → alt (a1, ..., an)) ∧ (a1 → p) ∧ ... ∧ (an → p)
- **Require**: a → b
- **Exclude**: a → ¬ b

FM can be used to determine the adaptation space of SAS, where each adaptation action is represented by a combination of features (Metzger et al., 2014).

Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning technique that consists of learning policies by trying actions in different situations and examining the results of each action (Sutton, 2014). Thanks to reinforcement learning, it is possible to extract knowledge about the managed subsystem (Ferreira et al., 2012). Figure 1 shown the RL process, which consists of an agent and an environment. The agent takes action ***a*** to be execute at time step ***t*** in environment state ***s***. Then the agent receives a reward ***r*** and the next state ***s'*** from the environment.

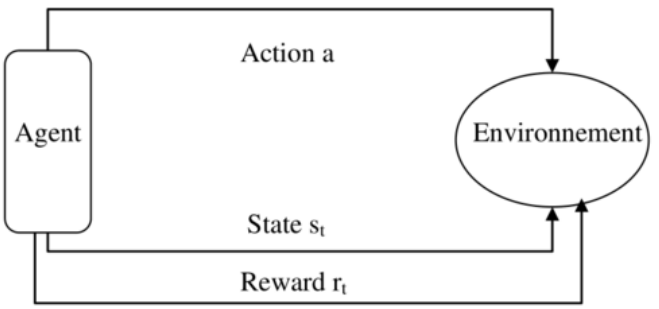
The main advantage of this method is that it does not require a model of the managed subsystem (Littman et al., 2004) (Dowling et al., 2006). However, it suffers from low scalability (Huebscher et al., 2008). To avoid this problem, the authors propose to integrate the exploration strategies that based on the FM into reinforcement learning algorithms. The most commonly used RL algorithms (Sutton et al., 2018) are Q-learning and SARSA (State-Action-Reward-State-Action). SARSA and Q-Learning use RL model-free algorithms. They are similar in exploitation strategies and differ in exploration strategies.

Q-learning (Wathins et al., 1992) is a model-free RL algorithm, proposed by Watkins. This algorithm provides a convenient way for an agent in order to learn from random action selection in order to ensure optimal performance. The goal of Q-learning is to estimate the quality value of action-state pairs. The learning quality function called Q represents an estimate of the accumulated reward that the algorithm obtains. The limitation of the Q-Learning algorithm is how to choose the appropriate action in a specific state. As a result, some policies (Yan et al., 2016) have appeared that facilitate the selection of the appropriate action, such as Boltzmann, ε-greedy, and greedy policy.

SAFRAN

SAFRAN (Self-Adaptive Fractal CompoNents) (David, 2005) is an extension of the Fractal component model aimed at supporting the development of self-adapting components. It supports logical structural adaptation and behavioral adaptation through parameterization and redirection of service invocations.

Figure 1. Reinforcement learning process (Sutton, 1998)



A Fractal (David et al., 2006) component has a membrane and a content. The membrane exposes interfaces listing operations, which are: server interface that provides operations, and client interface that requires operations. The membrane also exposes control interfaces. For example, act on the component's life cycle or its attributes, to introspect the component and also to intercept invocations of operations. Fractal supports vertical composition: the content of the component can encapsulate other components. It also supports the sharing of a component between several composites.

ADL Fractal (Bruneton et al., 2006) is an architecture description language that allows, as the name implies, to describe the architecture of a Fractal component regardless of its implementation.

The policies in SAFRAN are based on **E**vent-**C**ondition-**A**ction (ECA) rules. (1) The **E**vents use information sent by sensors independent of the system (these sensors are provided by the WildCat library). WildCat (David et al., 2005) is an extensible Java framework to ease the creation of context-aware applications. WildCAT provides a simple yet powerful dynamic model to represent an application's execution context. Its purpose is to facilitate the creation of context-aware applications. (2) The **C**ondition is specified with the dedicated FPath sub-language (David et al., 2009), which allows, among other things, to navigate in the architecture of the system and select the elements on which to apply the reconfigurations. (3) The **A**ctions are specified with the FScript language, making it possible to manipulate the components (modification of attribute values, addition of meta-components, etc.) and the architecture (connection / disconnection of components, addition/ deletion of sub-components ...). FScript (David et al., 2006) language enables the definition of complex reconfiguration scripts that can modify the structure and configuration of a Fractal application. FScript program consists of a set of function definitions and actions.

HCL_s BASED ON REINFORCEMENT LEARNING

Hierarchical Control Loops (HCLs)

HCLs (Ouareth et al., 2021) is an approach that ensures hierarchical control. Where, multiple control loops are structured as a hierarchical composition of MAPE entities. HCLs has emerged to achieve global adaptation without conflicts and to facilitate the separation of concerns using Fractal component model in the form of a hierarchy of MAPE loops. HCLs supports parameter and structure adaptation at runtime.

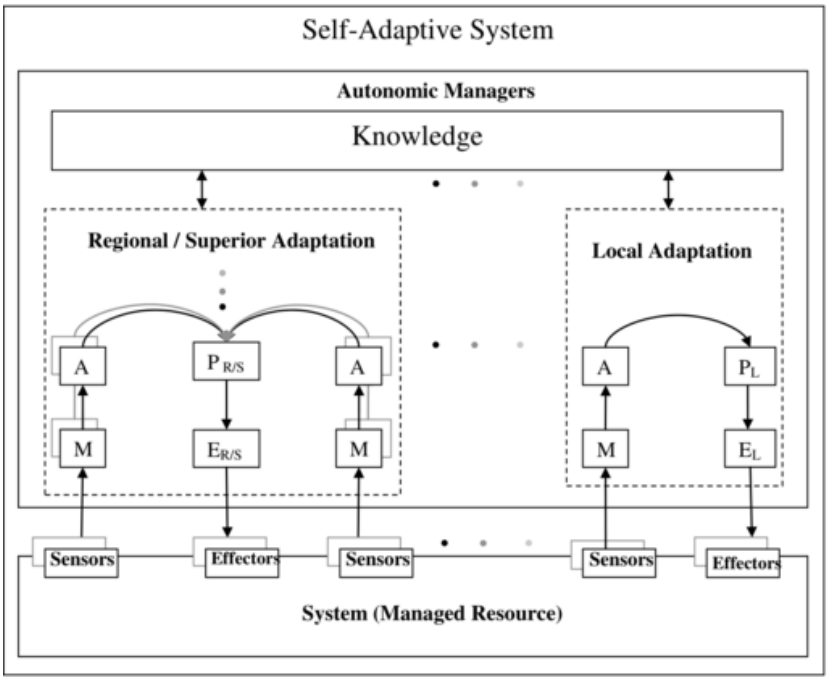
HCLs realizes the adaptation in three layers (see Figure 2) as follows:

- Local Adaptation (LA) that do not cross the boundaries of the adaptive component, where each adaptive component of the software can realize LA. This type supports parameter adaptation, i.e. must control and change only the parameter.
- Regional Adaptation (RA), where the region is determinate based on the architecture of the system. Only a component that contains at least one adaptive component can be considered a region.
- Superior Adaptation (SA) that crosses the boundaries of the region. SA allows adapting the component whatever their position in the hierarchy.

According to Ouareth et al. (Ouareth et al., 2020), each Monitor component operates independently from other Monitor component, as the purpose of monitor is to collect data and preprocess it to be in the standard format "symptoms", in order to simplify the analysis process. The generation of symptoms relies on a set of rules and strategies located in Knowledge base. This Knowledge base is shared between different Monitor components.

In the literature, some authors (Weyns et al., 2013) (Sylla et al., 2017) find it necessary to either centralize the component of the analysis or coordinate the components of the analysis if they are decentralized. However, the high cost of data transfer between coordinating entities may be an obstacle to achieving system scalability.

Figure 2. Hierarchical Control Loops pattern (HCLs) (Ouareth et al., 2021)



This paper focus on two problems related to: (i) how select the appropriate adaptation (i.e., LA, RA, SA), and (ii) how apply the coordination between Analyze entities in the case where it's necessary to put him.

Integration of RL Into the Component-Based MAPE-K Model

Figure 3 shows the integration of RL into the component-based MAPE-K reference model. Here, the authors have only modeled the RL process within the MAPE-K loop to clearly visualize the procedure. The basic functions of this loop are presented in (Ouareth et al., 2018) with many details.

The Exploration-Action component is responsible for exploring the adaptation action using Feature-Model and the knowledge received from the Analyze component. The Update-Policies components is responsible for updating policies that include in Adaptation-Policies and the action received from the Exploration-Action component.

Adaptation of Action Exploration Based on FM

Due to the evolutionary nature of SAS, the adaptation space is always in a dynamic state. Exploration strategies cross the FM to structure the adaptation space of the SAS. It helps to explore the next adaptation action. The adaptation action is represented by a valid feature combination that specifies the target runtime configuration of the system.

This article is based on exploring strategies to determine possible configurations on the autonomic manager. These strategies focus on adapting the coordination between the different "A" components. In addition, the selection of the appropriate type of adaptation is taken into consideration, to ensure correct adaptation without conflict and without high cost.

Figure 3. Integration of RL into the component-based MAPE-K model

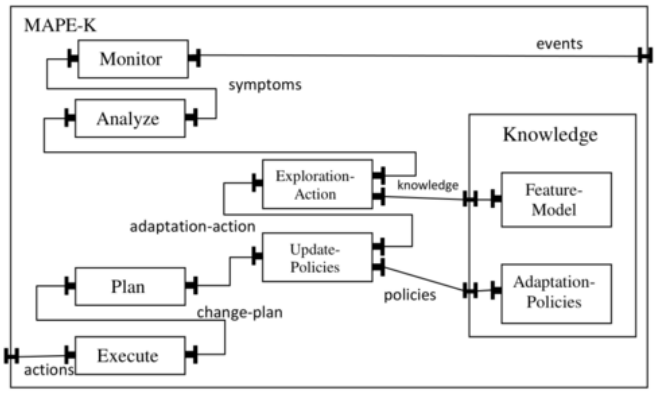
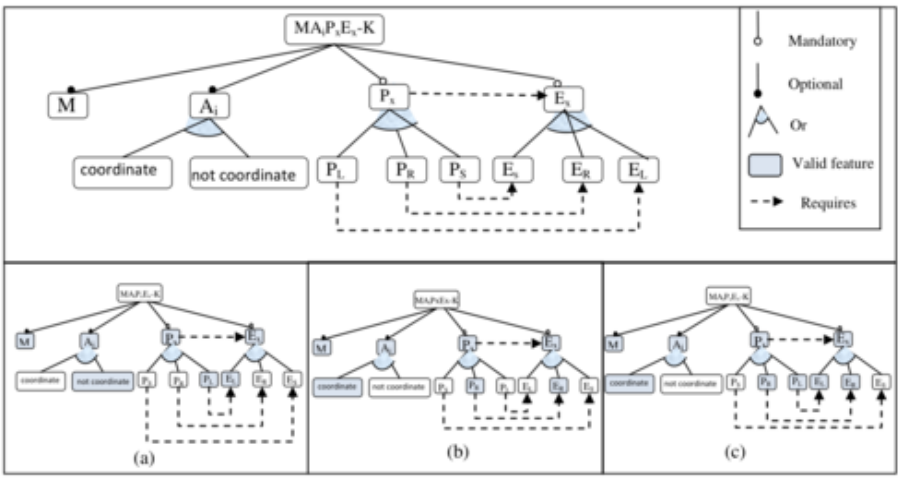


Figure 4 in the top part presents the FM of the MAPE-K control loop, where i Î ¥: represent the number of analyze (A) components existing in the system, and x Î {L, R, S} present the adaptation type to apply.

Feature M (i.e., Monitor) is mandatory and is always selected. The A_i (i.e., Analyze) feature is also always selected, and it has two sub-feature (coordinate and not coordinate), where you must select one of these two features. The feature "coordinate" means that component A_i must be coordinated with other(s) A component(s), and "not coordinate" means that component Ai does not need to be coordinated with another A component.

The P_x and E_x features are optional because sometimes the A component after their analysis finds that the detected state does not require adaptation. However, selecting the P_x feature requires selecting the E_x. Each P_x and Ex features has three sub-features (P_L, P_R, P_S) and (E_L, E_R, E_S) respectively.

Figure 4. Top: MAPE-K Feature Model, bottom: illustrative examples



The relation "Or" here means the obligation of selection of the P and E of the same type. For example, the selection of P_L implies the selection of E_L (see figure 4 (a)).

Figure 4 at the bottom part shows three examples (a, b, c) of the exploration of strategies, where the combination of features colored in gray gives an adaptation action. The exploration always starts with the root due to the existence of multiple MAPE-K control loops.

Exploration Strategies into Q-learning Algorithms

The proposed algorithm "Exploration Strategies into Q-learning" represents the extended algorithm of Q-learning. At the beginning, it is necessary to initialize the state of the system (see line 10) with an adaptation action a_0 (where, a_0 is the adaptation action determined at the design stage), which allows at the beginning to establish the coordination between all A components. Also, it allows applying the three types of adaptation. This step can be a reference to update knowledge on the basis of the best possible next action.

After initialization, the getExploratedAction function returns the next adaptation action (see line 13) repeatedly. After the selection of next action, the Q-learningMain procedure executes this action (line 14), then observes the reward (line 15). Finally, it allows updating the learning function Q and the state of the system S (line 16-17).

Exploration strategies are integrated into the Q-learning algorithm in the getExploratedAction function which selects the next adaptive action from the Autonomics Manager while making a comparison between exploration and exploitation. Authors based on the FM, and in particular on the relation between the different Features. FM is characterized by the root of the tree (parent) and a set of leaves (children). Each root has a set of children. The getExploratedAction function checks the relationship between root and child. If it is "Mandatory", the feature (child) will be added to the action. If the relation is optional, the decision to choose is random. In this case, if random()=1, the feature(child) will be added to the action, else this feature will not be added. The requireRelation function allows to check if this child is requested by another child. If so, it will be added to the action. If the relation between root and child is "Or", the randomSelectChild function returns one or more sub-feature. Finally, the getExploratedAction function stops when the last child is reached.

EXPERIMENT

In this section, the paper presents the case study and describes three scenarios to validate our proposition. The case study concerns an airport service providing wireless internet connection to passengers (Adamek et al., 2007) (Šerý et al., 2007). The objective of this system is to provide free internet access to passengers. Hanousse et al. (2011) are presented the component architecture of this application. According to Hanousse et al. (2011), there are three types of users, which are: (1) passengers use the number of their valid flight tickets to connect the network, (2) passengers use the number of their frequent flight tickets to connect the network with a permitted access time of the maximum provided time by the user flight tickets, and (3) regular users who have no tickets or who have already completed the time provided by the tickets. In this case, the user may create a temporary account and buy connection time.

- The **Arbitrator** component is the manager of the airport's wireless access system.
- The **InternetAccessManager** component is responsible for activating communication for the user.
- The **DhcpServer** component is responsible for connecting to the airport wireless system.
- The Token component is responsible for modeling a user session.
- The **WebAccountManager** component is responsible for creating a temporary account and buy access time.
- The **FlightTicketManager** component is responsible for managing flight tickets.
- The **FreqFlightTicketManager** component is responsible for managing the frequent flight tickets.

Algorithm

Algorithm 1: Exploration Strategies into Q-learning Algorithms

Require:

$S = \{s_0, s_1, s_2, ..., s_t\}$; ▷ State Space, where t represent discrete time step
$A = \{a_0, a_1, a_2, ..., a_n\}$; ▷ Adaptation action space, where $n \in N$
M ▷ Feature Model
$Q : S \times A \rightarrow \mathbf{R}$; ▷ Learn function
$T : S \times A \rightarrow S$; ▷ Function of state transition
$R : S \times A \rightarrow \mathbf{R}$; ▷ Reward function
$\alpha \in [0,1]$; ▷ Learning rate, typically α =0,1
$\gamma \in [0,1]$; ▷ Discounting factor

Procedure `Q-LearningMain(`M, S, A,α,γ`):`

- s_t = Intialize_FM(M,a_0, s_0); `// Initialize the space`
- **repeat**
  - root ← getRoot(M);
  - a_i=getExploratedAction(M,root); `// Selection of the Next Action`
  - $s_{t+1} \leftarrow T(s_t, a_i)$; `// Receive the new state`
  - $R_{t+1} \leftarrow R(s_t, a_i)$; `// Receive the reward`
  - $Q(s_t,a_i) \leftarrow Q(s_t,a_i)\ \alpha\ .\ [\ R_{t+1} + \gamma\ .\ \max_a Q(s_{t+1},a_i) - Q(s_t,a_i)]$; `// Update Q-factor associate to the state` s_t `and action` a_i
  - $s_t \leftarrow s_{t+1}$; `// Update the space`
- **until** *last time step*

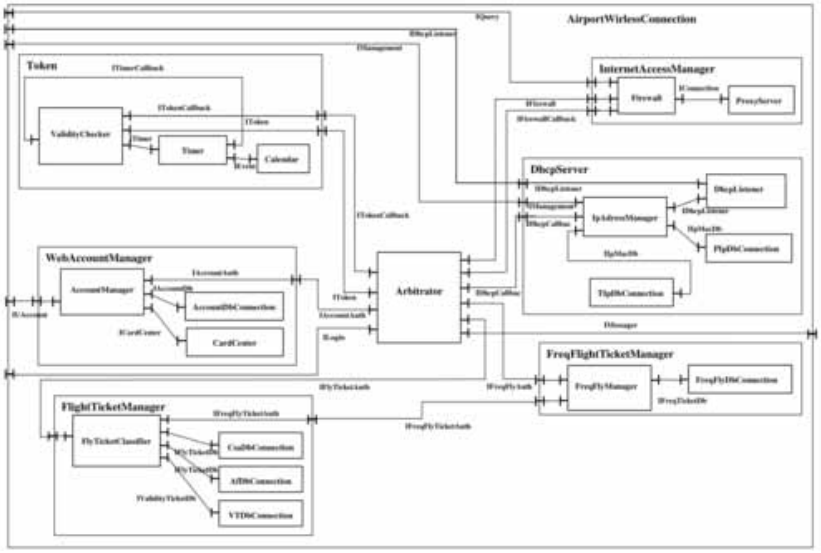
Function `Intialize_FM(`a, s`):`

- **for** i := *1 to* N **do**
  - `// N represent the number of control loops`
  - set ⟨ Feature ⟩ a ← { M, A_i, coordinate, P_X, P_L, P_R, P_S, E_X, E_L, E_R, E_S }; `// Initial action selection`
  - $s_{t+1} \leftarrow T(s_t,a)$; `// Receive the new state`
  - $s_t \leftarrow s_{t+1}$;
- **return** S_t

Function `getExploratedAction(`M,*root*`):`

- set ⟨ Feature ⟩ children ← getDescands(root);
- **for** *each child of the set of children* **do**
  - **if** *relation(root, child) = "Mandatory"* **then**
    - a ← addFeature(a, child);
  - **if** *relation(root, child) = "Optional"* **then**
    - **if** *random() = 1* **then**
      - a ← addFeature(a, child);
  - **if** *requireRelation(M, child)* $\neq$ " $\emptyset$ " *and (child* $\notin$ *a)* **then**
    - a ← addFeature(a, child);
  - **if** *relation(root, child) = "Or" and root* $\in$ *a and child* $\notin$ *a* **then**
    - child ← randomSeletChild(root);
    - a ← addFeature(a, child);
  - **if** *getDescands(child)* $\neq \emptyset$ **then**
    - a=getExploratedAction(M,child);
- **return** a

Figure 5. Fractal-based airport wireless access application



In this section, the authors add some modifications to the system architecture. Figure 5 shows the Fractal-based airport wireless access application.

Table 1 represents adaptation action space, where X means the feature is enabled. Exploration strategies into Q-Learning algorithms allow exploring strategies and determining the learning quality function Q in order to select possible configurations.

Table 1. Adaptation action space

Action	M	A	P_X			E_X		
			P_L	P_R	P_S	E_L	E_R	E_S
A	X	not coordinated	X			X		
B	X	coordinated	X			X		
C	X	not coordinated	X	X		X	X	
D	X	coordinated	X	X		X	X	
E	X	not coordinated	X	X	X	X	X	X
F	X	coordinated	X	X	X	X	X	X
G	X	not coordinated		X	X		X	X
H	X	coordinated		X	X		X	X
I	X	not coordinated			X			X
J	X	coordinated			X			X
K	X	not coordinated		X			X	
L	X	coordinated		X			X	
M	X	not coordinated	X		X	X		X
N	X	coordinated	X		X	X		X

IMPLEMENTATION

The objective is to introduce three scenarios that require self-adaptation in order to illustrate the validity of the proposed approach. The desired scenarios are:

Scenario 1: Local Adaptation

In this architecture, authors have added a component called "Calendar". This component is responsible for indicating all the events of the year. Whether national, international, or religious holidays.

The "Timer" component uses the provided interface (events) of the "Calendar" component to receive events (see Figure 5). The "ValidityChecker" component is responsible for starting the session "Timer" component and receiving information from the "Timer" component about the session time when elapses. On the other hand, the "ValidityChecker" in turn informs the Arbitrator.

On holidays, the system proposes to offer a bonus time to users according to their type. For example, 15 minutes for passengers with frequent flight tickets, 10 minutes for passengers with valid flight tickets, and 5 minutes for regular users. The "Timer" component also has an AttributeController interface of type TBonusAttribute, which provides two methods to get and set the attribute "TBonus" of the "Timer" component.

The "Timer" is an AC. When the "Timer" component receives an event, the M component collects the events and preprocesses them. The implementation of the M component must include the monitoring operation using WildCat, as shown in Figure 6.

The A component checks the rule and generates change requests. Then, the Exploration-Action and Update-Policies components apply the Q-learning algorithm based on the exploration strategies in order to find through experiences the learning quality function Q. After finding the Q values, the system selects the action that gives the biggest expected utility. Consequently, the adaptation action selected in this scenario is "A" (see table 1).

Next, the P_L component receives the change requests and determines the local change plan. The P_L component checks the type of client previously stored to decide what bonus time should be allocated. Figure 7 defines a new reconfiguration action named "adaptTimer", which takes two parameters: root is the signature of the "Token" component, and type, which represent the type of users.

Finally, The E_L component resets the timer for the corresponding bonus time. Figure 8 shows the self-adaptation of the timer for different types of users.

Scenario 2: Regional Adaptation

The airport wireless connection system offers the advantage of protecting minors (age <= 19) from the dangers of the Internet. The "Firewall" component is an AC, and works to block unauthorized internet connections. On the other hand, it sends enabled IP addresses to the "ProxyServer". Once

Figure 6. LA scenario: WildCat definition to sensor events

```
Context ctx = new Context();
DynamicContextProvider provider = ctx.createDynamicContextProvider();
ctx.register(EventKind.CONDITION_OCCURED, createPath("/AirportWirlessConnection/Token/*#*"),
        new LoggingContextListener(root, "Timer"));
try {
    ctx.mount("/AirportWirlessConnection/Token", provider);
} catch (InvalidMountPointException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
provider.createResource(createPath("Timer"));
provider.attachSensor(createPath("Timer"), new EventSensor(root,"Timer"),
        8000, MILLISECONDS);
provider.startSensor(createPath("Timer"));
```

Figure 7. LA scenario: FScript definition

```
action adaptTimer(root, type) {

       Timer = $root/descendant::Timer;
       t = value($Timer/attribute::t);

       if($type == "FreqFly") {
        t = $t + 15;
        }
        if($type == "ValidFly") {
        t = $t + 10;
        } else {
        t = $t + 5;
        }
        set-value($Timer/attribute::t, $t);
        }
```

Figure 8. LA scenario

users are connected, it sends queries to the "InternetAccessManager" (use IQuery interface). The "InternetAccessManager" component forwards users' requests to the "Arbitrator". To access user information, for example, the age of the user, the "Firewall" sends a request to the "Arbitrator" component, using the IFirewall interface. The "Arbitrator" sends request to the "FlightTicketManager", "FreqFlightTicketManager", and "WebAccountManager" components using the IFlyTicketAuth, IFreqFlyAuth, and IAccoutAuth interfaces, respectively.

The "Firewall" receives user information from the Arbitrator component using IFirewallCallback interface. The "Filter" component is optional and only appears when the user is minor. The

"Firewall" sends the requested IP connection to the "Filter" component in order to check whether the requested IP connection is in the black list or not. Figure 9 shows a Fractal ADL description of the "InternetAccessManager" CC. It contains two sub-components: "Firewall" and "ProxyServer" before the application of adaptation.

When the "Firewall" is received a queries on the IQuery interface, the sensors capture it, then the M component collected the captured data. In addition, it preprocesses and generates symptoms. Figure 10 shows the implementation of the M component using WildCat.

The A component receives the symptoms and analyzed them according to ECA rules (policies in SAFRAN). It checks if the "Filter" component is not already added, if so, it must added the "Filter" component into the "InternetAccessManager" composite (see Figure 11). Then, the Exploration-Action and Update-Policies components apply the Q-learning algorithm based on the exploration strategies in order to find through experiences the learning quality function Q. After finding the Q values, the system selects the action that gives the biggest expected utility. Consequently, the adaptation action selected in this scenario is "K" (see table 1).

Figure 9. ADL Fractal description of the InternetAcessManager CC

```
<definition name="AirportW.InternetAccessManager">

  <interface name="IQuery" role="server" signature="AirportW.IQuery"/>
  <interface name="IFirewall" role="server" signature="AirportW.IFirewall"/>
  <interface name="IFirewallCallback" role="server" signature="AirportW.IFirewallCallback"/>

  <component name="Firewall" definition="AirportW.Firewall" />
  <component name="ProxyServer" definition="AirportW.ProxyServer" />

  <binding client="this.IQuery" server="Firewall.IQuery" />
  <binding client="this.IFirewall" server="Firewall.IFirewall" />
  <binding client="this.IFirewallCallback" server="Firewall.IFirewallCallback" />
  <binding client="Firewall.IConnection" server="ProxyServer.IConnection" />

</definition>
```

Figure 10. WildCat definition to sensor the age of users

```
Context ctx = new Context();
DynamicContextProvider provider = ctx.createDynamicContextProvider();
ctx.register(EventKind.CONDITION_OCCURED, createPath("/AirportWirlessConnection/InternetAccessManager/*"),
        new LoggingContextListener(root, "Firewall"));
try {
    ctx.mount("/AirportWirlessConnection/InternetAccessManager", provider);
} catch (InvalidMountPointException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
provider.createResource(createPath("Firewall"));
provider.attachSensor(createPath("Firewall"), new ageSensor(root,"Firewall"),
        8000, MILLISECONDS);
provider.startSensor(createPath("Firewall"));
```

Figure 11. RA scenario: Adaptation policy definition

```
rule{
when realized($root/descendant::Firewall/attribute::age <= 19)
if (not($root/descendant::Filter))
do {
add_Filter_component($root);
}
}
```

Next, the P_R component receives the change requests and determines the regional change plan which contains the following actions:

add_component(InternetAccessManager, Filter): This action allows to add a new sub-component (Filter) to a (InternetAccessManager) CC.

bind(IFilter, IFilter): The bind action allows to connect the required interface with the provided interface.

Figure 12 defines a new reconfiguration action named "add_Filter_component", which takes a parameter "root", which is the signature of the "InternetAccessManager" component.

Finally, The E_R component controls the execution of the adaptive action "add_Filter_component". Figure 13 shows the self-adaptation of the structure of the "InternetAccessManager" CC.

Figure 14 shows the Fractal-based "InternetAccessManager" CC after the adaptation application.

Figure 12. RA scenario: FScript definition

```
action add_Filter_component(root) {

            filter = new('AirportW.Filter');
            set-name($filter, 'Filter');
            add($root, $filter);
            fire-wall = $root/descendant::Firewall;
            stop($root);
            bind($filter/interface::IFilter, $fire-wall/interface::IFilter);
            start($root);

        }
```

Figure 13. RA scenario

Scenario 3: Superior Adaptation

The "CustomToken" component is optional and only appears when regular users create an account. The "AccountManager" component is an AC, and it works to update the permitted connection time of the user in the "WebAccountManager". Regular users can create a temporary account (Managed by "AccountManager" component), and buy for an indicated access time using IUAccount interface of the "AirportWirelessConnection" component. The "AccountDbConnection" is responsible for managing the consumed time. Figure 15 shows a Fractal ADL description of the "Token" CC. It contains two sub-components: "Firewall" and "ProxyServer" before the application of adaptation.

When the "AccountManager" is received a request to create a temporary account, the sensors capture it, then the M component collected the captured data. In addition, it preprocesses and generates symptoms.

The A component receives the symptoms and analyzed them according to ECA rules (policies in SAFRAN). It checks if the "CustomToken" component is not already added, if so, it must added the "CustomToken" component into the "WebAccountManager" region (see Figure 17). Then, the Exploration-Action and Update-Policies components apply the Q-learning algorithm based on the exploration strategies in order to find through experiences the learning quality function Q. After finding the Q values, the system selects the action that gives the biggest expected utility. Consequently, the adaptation action selected in this scenario is "I" (see table 1).

Figure 14. Fractal-based InternetAcessManager CC after the adaptation application

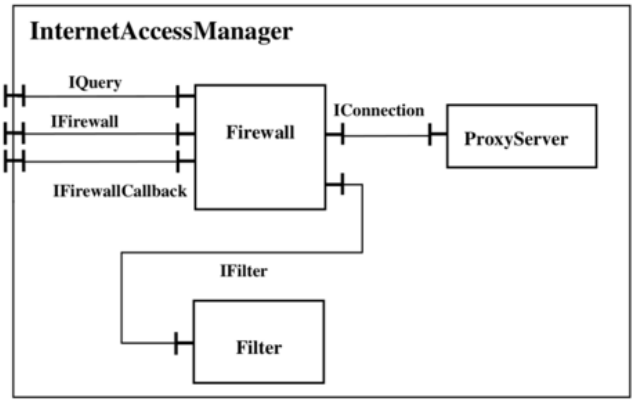


Figure 15. ADL Fractal description of the Token CC

```
<definition name="AirportW.Token">
  <interface name="IToken" role="client" signature="AirportW.IToken"/>
  <interface name="ITokenCallback" role="client" signature="AirportW.ITokenCallback"/>

  <component name="ValidityChecker" definition="AirportW.ValidityChecker" />
  <component name="Timer" definition="AirportW.Timer" />
  <component name="Calendar" definition="AirportW.Calendar" />

  <binding client="ValidityChecker.ITokenCallback" server="this.ITokenCallback" />
  <binding client="ValidityChecker.IToken" server="this.IToken" />
  <binding client="Timer.ITimerCallback" server="ValidityChecker.ITimerCallback" />
  <binding client="Timer.IEvent" server="Calendar.IEvent" />
  <binding client="ValidityChecker.ITimer" server="Timer.ITimer" />

</definition>
```

Figure 16. WildCat definition to sensor the creation of the account

```
Context ctx = new Context();
DynamicContextProvider provider = ctx.createDynamicContextProvider();
ctx.register(EventKind.CONDITION_OCCURED, createPath("/AirportWirlessConnection/WebAccountManager/*#*"),
        new LoggingContextListener(root, "AccountManager"));
try {
    ctx.mount("/AirportWirlessConnection/Token", provider);
} catch (InvalidMountPointException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
provider.createResource(createPath("AccountManager"));
provider.attachSensor(createPath("AccountManager"), new AccountSensor(root,"AccountManager"),
        8000, MILLISECONDS);
provider.startSensor(createPath("AccountManager"));
```

Next, the P_s component receives the change requests and determines the superior change plan which contains the following actions:

add_component(Token, CustomToken): This action allows to add a new sub-component (CustomToken) to a CC (Token).

bind(ICustomCallback, ICustomCallback) **bind**(IAccount, IAccount): The bind action allows to connect the required interface with the provided interface.

Figure 18 defines a new reconfiguration action named "add_CustomToken_component", which takes a parameter "root", which is the signature of the "AirportWirlessConnection" component.

Finally, The E_s component controls the execution of the adaptive action "add_CustomToken_component". Figure 19 shows the self-adaptation of the structure of the "AirportWirlessConnection" CC, and precisely, the "Token" and "WebAccountManager" CC.

Figure 20 shows the Fractal architecture of the "Token" and "WebAccountManager" CC after the adaptation application.

Figure 17. SA scenario: Adaptation policy definition

```
rule{
when realized($root/descendant::WebAccountManager/descendant::AccountManager
                                                /attribute::newAccount == true)
if (not($root/descendant::Token/descendant::CustomToken)
do {
add_CustomToken_component($root);
}
}
```

Figure 18. SA scenario: FScript definition

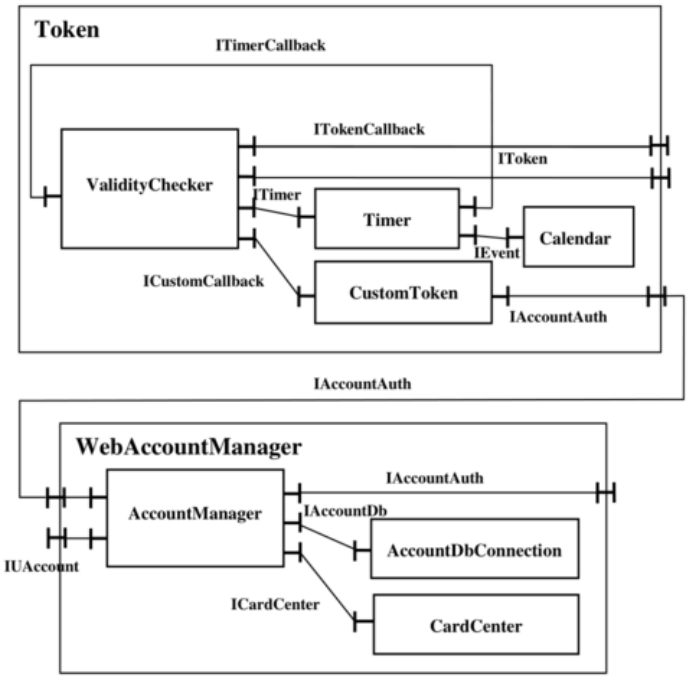
```
action add_CustomToken_component(root) {

        token=$root/descendant::Token;
        customToken = new('AirportW.CustomToken');
        set-name($customToken, 'CustomToken');
        add($token, $customToken);

        validityChecker = $token/descendant::ValidityChecker;
        webAccountManager= $root/descendant::WebAccountManager;
        accountManager= $webAccountManager/descendant::AccountManager;
        stop($root);
        bind($validityChecker/interface::ICustomCallback, $customToken/interface::ICustomCallback);
        bind($customToken/interface::IAccountAuth, $token/interface::IAccountAuth);
        bind($token/interface::IAccountAuth, $webAccountManager/interface::IAccountAuth);
        bind($webAccountManager/interface::IAccountAuth, $accountManager/interface::IAccountAuth);
        start($root);
    }
```

Figure 19. SA scenario

Figure 20. Fractal-based of the Token and WebAccountManager CC after the adaptation application



RELATED WORK

Recently, the researchers focused on the use of RL to support self-adaptation in several studies. Quin et al. (2019) proposed an approach based on RL. The authors propose to improve the control loop by integrating a learning module in MAPE-K. The objective of the learning module is to select subset of adaptation option from a large adaptation space in order to ensure effective performance of the Analyze module. Metzger et al. (2022) proposed to integrate the exploration strategies into reinforcement learning algorithms. These strategies are based on the feature model that represents the structure of the system. The adaptation action is represented by a combination of a set of valid features. Weyns et al. (2022) proposed DLASeR+ (Deep Learning for Adaptation Space Reduction Plus) based on RL in order to reduce large adaptation spaces and classify adaptation options efficiently way with threshold, optimization, and set-point goals.

In related works, everyone focuses on the use of RL in the following cases: reducing the adaptation space, seeking efficient decision-making, and exploring a large adaptation space. But, only the proposed approach focuses on exploring strategies to adapt the autonomic managers. The advantage is to reduce the cost of coordination thanks to the possibility of adapting the structure and the behavior of the autonomic managers. As a result, this contribution ensures correct and consistent adaptation without conflicts.

CONCLUSION

In this paper, the authors extend the model of HCLs. Where, the authors relied on exploration strategies for online RL, using the feature model to define an adaptation space. The goal is the capability of changing of the structure and behavior of loops at runtime. Also, to avoid the conflicting decision, and to reduce the cost of coordination.

The HCLs have been evaluated in a wireless connection domain for passengers at the airport. The results of the evaluation demonstrate the feasibility and validity of HCLs using SAFRAN. This evaluation presented the following advantages: (i) a pattern that enable to achieve global adaptation without conflict, (ii) facilitates the separation of concerns and reconfiguration at runtime thanks to SAFRAN, and (iii) composition without complex coordination.

The limitations of this proposed approach lie in the difficulty of its implementation due to the multiplicity of languages used in programming. Therefore, it is necessary to find or suggest a means that facilitates the process of applying this approach so that it becomes practical and easy to develop and simulate the approach.

In future efforts, the authors plan to implement HCLs in another execution domain. Also, the authors will address the problem of uncertainty in SAS issue at scale.

REFERENCES

Adamek, J., Bures, T., Jezek, P., Kofron, J., Mencl, V., Parizek, P., & Plasil, F. (2007). *Component reliability extensions for fractal component model*. Academy of Sciences of the Czech Republic and France Telecom.

Al-Rahmi, W. M., Othman, M. S., Yusof, L. M., & Musa, M. A. (2015). Using social media as a tool for improving academic performance through collaborative learning in Malaysian higher education. *Review of European Studies*, *7*, 265.

Arcaini, P., Gargantini, A., & Vavassori, P. (2015, April). Generating tests for detecting faults in feature models. In *IEEE 8th International Conference on Software Testing, Verification and Validation (ICST),* (pp. 1-10). IEEE.

Barna, C., Ghanbari, H., Litoiu, M., & Shtern, M. (2009). Software engineering for self-adaptive systems.

Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., & Stefani, J. B. (2006). The fractal component model and its support in java. *Software, Practice & Experience*, *36*(11-12), 1257–1284.

Calinescu, R., Gerasimou, S., & Banks, A. (2015, April). Self-adaptive software with decentralised control loops. In *International Conference on Fundamental Approaches To Software Engineering,* (pp. 235-251). Springer, Berlin, Heidelberg. doi:10.1007/978-3-662-46675-9_16

Chang, V., Abdel-Basset, M., & Ramachandran, M. (2019). Towards a reuse strategic decision pattern framework–from theories to practices. *Information Systems Frontiers*, *21*(1), 27–44.

David, P. C. (2005). Développement de composants Fractal adaptatifs: un langage dédié à l'aspect d'adaptation. [Doctoral dissertation.] Université de Nantes.

David, P. C., & Ledoux, T. (2005, November). WildCAT: a generic framework for context-aware applications. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, (pp. 1-7).

David, P. C., & Ledoux, T. (2006, July). Safe dynamic reconfigurations of fractal architectures with fscript. In *Proceeding of Fractal CBSE Workshop, ECOOP* (Vol. 6).

David, P. C., Ledoux, T., Léger, M., & Coupaye, T. (2009). FPath and FScript: Language support for navigation and reliable reconfiguration of Fractal architectures. *annals of telecommunications-annales des télécommunications, 64*(1), 45-63.

Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., & Zambonelli, F. (2006). A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, *1*(2), 223–259. doi:10.1145/1186778.1186782

Dowling, J., Cunningham, R., Curran, E., & Cahill, V. (2006). Building autonomic systems using collaborative reinforcement learning. *The Knowledge Engineering Review*, *21*(3), 231–238.

Ferreira, J., Leitão, J., & Rodrigues, L. (2012). A-OSGi: A framework to support the construction of autonomic OSGi-based applications. *International Journal of Autonomous and Adaptive Communications Systems*, *5*(3), 292–310.

Hachicha, M., Ben Halima, R., & Hadj Kacem, A. (2017, December). Designing compound MAPE patterns for self-adaptive systems. In *International Conference on Intelligent Systems Design and Applications,* (pp. 92-101). Springer, Cham.

Hannousse, A. (2011). Aspectualizing component models: implementation and interferences analysis. [Doctoral dissertation.] Ecole des Mines de Nantes.

Huebscher, M. C., & McCann, J. A. (2008). A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys*, *40*(3), 1–28.

Littman, M. L., Ravi, N., Fenson, E., & Howard, R. (2004, May). Reinforcement learning for autonomic network repair. In *International Conference on Autonomic Computing, 2004. Proceedings,* (pp. 284-285). IEEE.

Metzger, A., & Pohl, K. (2014). Software product line engineering and variability management: achievements and challenges. In Future of software engineering proceedings, (pp. 70-84).

Metzger, A., Quinton, C., Mann, Z. Á., Baresi, L., & Pohl, K. (2022). Realizing self-adaptive systems via online reinforcement learning and feature-model-guided exploration. *Computing*, 1–22.

Ouareth, S., Boulehouache, S., & Mazouzi, S. (2018, October). A component-based mape-k control loop model for self-adaptation. In *3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)*, (pp. 1-7). IEEE.

Ouareth, S., Boulehouache, S., & Mazouzi, S. (2021, December). An Approach for Composing Multiple Control Loops Hierarchically. In *International Conference on Theoretical and Applicative Aspects of Computer Science (ICTAACS),* (pp. 1-5). IEEE.

Ouareth, S., Boulehouache, S., & Smaine, M. (2020). Reliable Composition of MAPE-K Loops in Self-Adaptive Software Systems. *International Journal of Organizational and Collective Intelligence*, *10*(3), 38–54.

Quin, F., Weyns, D., Bamelis, T., Buttar, S. S., & Michiels, S. (2019, May). Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In *IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS*), (pp. 1-12). IEEE.

Šerý, O., & Plášil, F. (2007, July). Slicing of component behavior specification with respect to their composition. In *International Symposium on Component-Based Software Engineering,* (pp. 189-202). Springer, Berlin, Heidelberg.

Sutton, R. S. (1998, November). Reinforcement learning: Past, present and future. In *Asia-Pacific Conference on Simulated Evolution and Learning,* (pp. 195-197). Springer, Berlin, Heidelberg.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sylla, A. N., Louvel, M., Rutten, E., & Delaval, G. (2017, September). Design framework for reliable multiple autonomic loops in smart environments. In *International conference on cloud and autonomic computing (ICCAC),* (pp. 131-142). IEEE.

Vromant, P., Weyns, D., Malek, S., & Andersson, J. (2011, May). On interacting control loops in self-adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems,* (pp. 202-207). doi:10.1145/1988008.1988037

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3), 279–292.

Weyns, D., Gheibi, O., Quin, F., & Donckt, J. V. D. (2022). *Deep Learning for Effective and Efficient Reduction of Large Adaptation Spaces in Self-Adaptive Systems. ACM Transactions on Autonomous and Adaptive Systems*. TAAS.

Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., & Göschka, K. M. (2013). On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II,* (pp. 76–107). Springer. doi:10.1007/978-3-642-35813-5_4

Yan, J., He, H., Zhong, X., & Tang, Y. (2016). Q-learning-based vulnerability analysis of smart grid against sequential topology attacks. *IEEE Transactions on Information Forensics and Security*, *12*(1), 200–210.

Zavala, E., Franch, X., Marco, J., & Berger, C. (2020). HAFLoop: An architecture for supporting Highly Adaptive Feedback Loops in self-adaptive systems. *Future Generation Computer Systems*, *105*, 607–630.

Selma Ouareth is a PhD student at 20 Août 1955-Skikda University, Algeria. She is a member of the Autonomic and Intelligent System team of the LICUS laboratory at 20 Août 1955-Skikda University since 2020. Currently, she works in the field of self-adaptive systems based on multiple control loops. His research interests focus on software architecture and the engineering of Artificial Intelligence for such complex system. s.ouareth@gmail.com

Dr Soufiane Boulehouache received his Master and PhD at Abdelhamid Mehri Constantine 2 (Constantine University), Algeria. He is a former member of GLIA research team of the LIRE laboratory at Abdelhamid Mehri Constantine 2 University, Algeria. Also, he was the founder and the head of Distributed Computer Systems and Networks Bachelor's Specialty. He was the head of Computer Science department twice. Actually, at the same time, he is a Senior Lecturer "Classe A" and the head of the Autonomic and Intelligent System research team of the LICUS laboratory at 20 Août 1955-Skikda University, Algeria. Also, he is the head of the Computer Science Field (Responsable de spécialité). In addition, he is a supervisor of many PhD students. His research interests include the use of Artificial Intelligence and Software Engineering methods and tools to engineer Self-Adaptive Systems in general and Pedagogical Systems in particular. s.boulehouache@univ-skikda.dz

Smaine Mazouzi received his M.S and Ph.D. degrees in computer science from university of Constantine, respectively in 1996, and 2008. He is a professor at 20 Août 1955 university of Skikda and the head of the team Artificial intelligence of the LICUS laboratory (Laboratoire d'Informatique et de Communication de l'Université de Skikda), at the same university. His fields of interest are pattern recognition, machine vision, distributed systems, and computer security. His current research concerns using distributed and complex systems modeled as multi-agent systems in image understanding and intrusion detection. He is member of several national and international research projects in computer vision and computer security. mazouzi_smaine@yahoo.fr