Future Generation Computer Systems 105 (2020) 607–630

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

HAFLoop: An architecture for supporting Highly Adaptive Feedback Loops in self-adaptive systems

Edith Zavala [a,*], Xavier Franch [a], Jordi Marco [b], Christian Berger [c]

[a] Service and Information System Engineering Department, Polytechnic University of Catalonia, Jordi Girona 1-3, 08030, Barcelona, Spain
[b] Computer Science Department, Polytechnic University of Catalonia, Jordi Girona 1-3, 08030, Barcelona, Spain
[c] Department of Computer Science and Engineering, Chalmers and the University of Gothenburg, Hörselgången 5, 41296 Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 13 July 2019
Received in revised form 7 December 2019
Accepted 17 December 2019
Available online 24 December 2019

Keywords:
Self-adaptive system
Smart vehicle
IoT system
Adaptive monitoring
Adaptive feedback loop
Self-improvement

ABSTRACT

Most of the current self-adaptive systems (SASs) rely on static feedback loops such as the IBM's MAPE-K loop for managing their adaptation process. Static loops do not allow SASs to react to runtime events such as changing adaptation requirements or MAPE-K elements' faults. In order to address this issue, some solutions have emerged for manually or automatically perform changes on SASs' feedback loops. However, from the software engineering perspective, most of the proposals cannot be reused or extended by other SASs. In this paper, we present HAFLoop (Highly Adaptive Feedback control Loop), a generic architectural proposal that aims at easing and fastening the design and implementation of adaptive feedback loops in modern SASs. Our solution enables both structural and parameter adaptation of the loop elements. Moreover, it provides a highly modular design that allows SASs' owners to support a variety of feedback loop settings from centralized to fully decentralized. In this work, HAFLoop has been implemented as a framework for Java-based systems and evaluated in two emerging software application domains: self-driving vehicles and IoT networks. Results demonstrate that our proposal easies and accelerates the development of adaptive feedback loops as well as how it could help to address some of the most relevant challenges of self-driving vehicles and IoT applications. Concretely, HAFLoop has demonstrated to improve SASs' feedback loops' runtime availability and operation.

1. Introduction

Self-adaptive systems (SASs) like smart cities, smart vehicles and mobile applications, have been subject of considerable research effort in the last years [1–4]. A SAS is a system able to automatically modify itself in order to respond to changes in its environment and the system itself [2]. This kind of systems is composed of an Autonomic Manager (AM) (also referred to as Adaptation Logic) and the Managed Elements (MEs) (also referred to as Managed Resources) [1,5]. The MEs are the components of the system that provide the main functionality and can be adapted while the AM corresponds to the control unit that manages the MEs' adaptation process [1]. In practice, the AM is typically implemented through a feedback control loop. One of the most popular feedback loops is the IBM's MAPE-K loop [6,7]. This loop consists of five elements: Monitor, Analyze, Plan, Execute and a Knowledge base. In addition, it includes two interface elements for interacting with the MEs: *Sensors* and *Effectors* (see Fig. 1). According to IBM's proposal [6]:

- The **Monitor** element is in charge of providing the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from MEs using the *Sensors* interface.
- The **Analyze** element provides the mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models). These mechanisms allow the AM to understand about the IT environment and help to predict future situations.
- The **Plan** element provides the mechanisms that construct the actions needed to achieve goals and objectives. It uses policy information to guide its task.
- The **Execute** element is in charge of providing the mechanisms that control the execution of a plan with considerations for dynamic updates using the *Effectors* interface.
- Finally, the **Knowledge base** element manages knowledge sources that can contain different types of knowledge (for example, data resulting from elements' operation and management data) for supporting the operation of the loop.

Most of the current approaches supporting SASs focus on the adaptation of the MEs and consider the elements of the loop static

* Corresponding author.
E-mail addresses: zavala@essi.upc.edu (E. Zavala), franch@essi.upc.edu (X. Franch), jmarco@cs.upc.edu (J. Marco), christian.berger@gu.se (C. Berger).

https://doi.org/10.1016/j.future.2019.12.026
0167-739X/

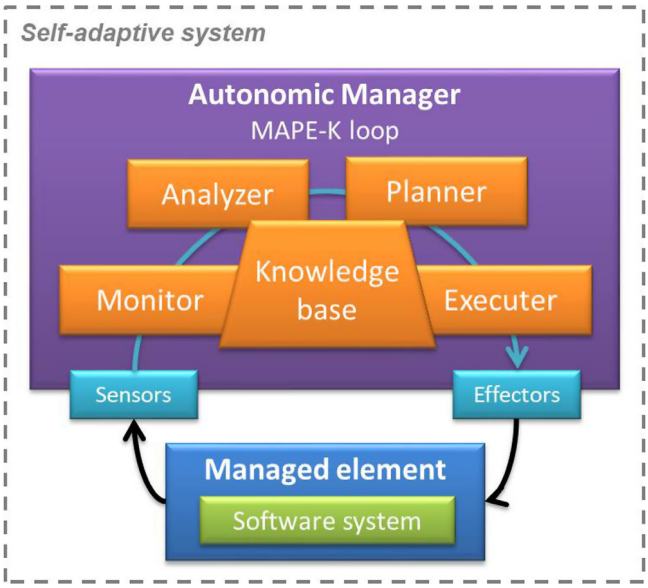


Fig. 1. MAPE-K loop.

components, i.e., the MAPE-K elements are not able to change their structure or behavior at runtime. However, the adaptation of these elements might be beneficial and even necessary in some situations, e.g., to respond to changes in the system resources, its environment or its adaptation goals and requirements, as well as to deal with unanticipated events such as AM elements' faults [8–12]. For instance, consider the case of a self-driving vehicle (SDV) with adaptation capabilities. In this type of system, the Monitor element would manage a set of sensors for gathering contextual data (e.g., from the driver and the environment) which would be later analyzed for making adaptation decisions (e.g., change the driving mode or turn on an alarm). In this example, a runtime faulty sensor may prevent the vehicle to correctly satisfy adaptation requirements, e.g., change to a higher autonomous level on a highway.

The goals of adapting the elements that compose the AM, i.e., the MAPE-K elements, can have different purposes. For example, to satisfy a certain quality level such as quality of service or output quality, or to reduce communication overhead, i.e., messages sent between MAPE-K elements on the same or remote machines [13]. Other purposes could be: to increase performance, to respond to components' faults, or to provide just-in-time adaptation. These goals can be achieved by structural adaptation (e.g., replacing one software component by another) or by parameter adaptation (e.g., changing the frequency of the loop's iterations). Moreover, the identification of the need for adaptation can be proactive (before an event happens, e.g., before systems' battery depletes) or reactive (after an event happens, e.g., once a failure occurs) [1].

From the state-of-the-art studies analyzed in this work, we have identified some approaches that focus on the adaptation of SASs' AM components. However, they present some limitations, e.g., they are so specific that cannot be reused by other SASs, they cannot support different types of adaptation. Moreover, guidelines for adopting such proposals in decentralized environments, such as the ones that accommodate modern SASs, are missing in the majority of the cases. Motivated by these facts, in this work, we propose HAFLoop (Highly Adaptive Feedback control Loop), an architectural solution that extends the MAPE-K loop reference model for enabling the adaptation of its elements, at runtime. HAFLoop proposes a generic structure for the elements as well as the mechanisms required for coordinating their operation with their adaptation process. Our solution can be reused and extended by other approaches for easing and accelerating the development of adaptive feedback loops as well as for supporting a variety of settings, from centralized to fully decentralized loops.

Among the MAPE-K loop elements, the Monitor element plays a crucial role since the quality of the monitored data (i.e., availability, timeliness, freshness, accuracy, etc.) impacts directly the performance of the rest of the elements of the loop, as exemplified before by the SDV scenario. In a systematic study that we have recently performed on the adaptive monitoring topic [14], we have found only very few approaches supporting adaptive monitoring in SASs. Therefore, in this work, we evaluate HAFLoop focusing on providing adaptation capabilities to the Monitor element of the loop. The evaluation is conducted on two SASs' application domains: SDVs and IoT networks. Adaptation of the Monitor element is enabled in both domains for responding to monitors' runtime faults, limited resources and degraded performance. In order to conduct the evaluation, we have utilized two simulation environments: the open-source middleware OpenDaVINCI [15], for the SDV use case; and, the DeltaIoT [16] artifact exemplar, for the IoT Network use case.

The remainder of the paper is organized as follows. Section 2 introduces a running example that motivates and later illustrates the main concepts of our proposal. Section 3 provides the background of this work. Section 4 presents the work related to our proposal. Section 5 describes our proposal, HAFLoop, while Section 6 presents the settings and results of its evaluation. Finally, Section 7 draws the conclusions and future work.

2. Running example

Aligned with the evaluation that we will present later in this work, we use as a running example an SDV with adaptation capabilities, i.e., a smart SDV (see Fig. 2). Smart (or intelligent) vehicles are systems capable of sensing data (e.g., from the driver, the environment, the vehicle itself) and making decisions based on these data (e.g., turn on an alarm, change driving route, activate self-driving functionality) [17], i.e., self-adapt. These systems have become increasingly popular in the automotive industry. Smart vehicles are bringing several societal benefits, for instance, improving drivers' safety, optimizing fuel consumption and improving driver's experience and comfort. In consequence, the interest of the research community in this domain has increased steadily. At the same time, their control systems need to face challenging runtime factors such as limited resources and unpredictable events what make this domain still subject of research.

Many researchers have focused on studying the self-driving functionality of smart vehicles and allowing drivers to concentrate on other tasks, from working to relaxing [15]. However, less effort has been done on studying how runtime factors affect to, and could be addressed in, this kind of system. In our example, we consider a hybrid SDV driving a user from work to home in a daily basis. In this daily journey, several runtime events could affect the performance of the self-driving functionality, for instance, running out of battery or a sensor fault. We take these two factors as example scenarios that our SDV is able to cope through the adaptation of the Monitor element of the AM (i.e., the Monitoring strategy in Fig. 2). In the first case, battery issues, our solution prolongs the availability of the self-driving functionality by performing a trade-off between sensors' utility and their energy consumption as well as the frequency at which they gather data. In the second case, a sensor fault, the adaptation of the Monitor element consists of the activation of alternative sensors, maintaining in this way the required confidence level.

In order to correctly support the scenarios mentioned above, we have to ensure that: (1) the Monitor element of the loop

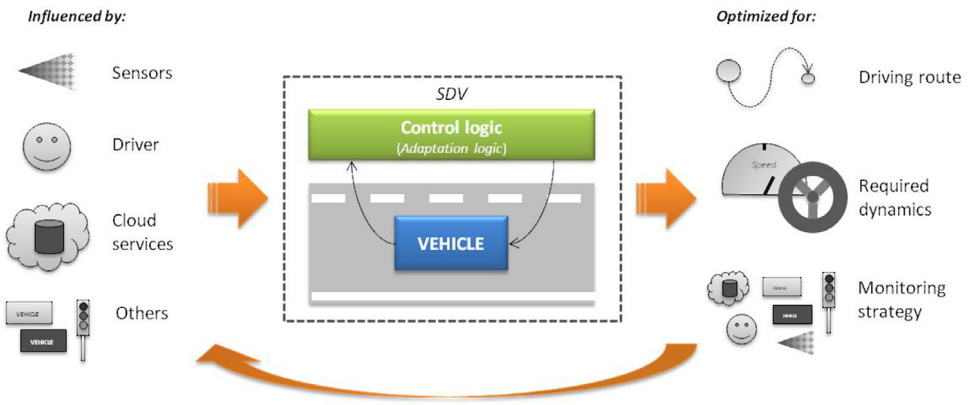


Fig. 2. Self-driving vehicle with adaptation capabilities.

(operating at the *Adaptation logic*, see Fig. 2) correctly enacts the adaptation required; (2) the adaptation is enacted timely. For satisfying the first requirement, our solution must correctly identify the need for the adaptation, analyze the context, and decide the changes to be enacted on the vehicle's monitoring. For satisfying the second requirement, the process of identifying, analyzing and executing the adaptation, should be performed in a period of time that enables the SDV to correctly support the self-driving functionality, i.e., ensure its availability. In the evaluation of our proposal, we demonstrate how these scenarios can be addressed adopting HAFLoop.

3. Background

Before describing HAFLoop in detail, we introduce in this section the background knowledge necessary to understand our proposal. It is divided into two main parts: a software engineering pattern that guides the structural aspects of HAFLoop; and, a generic template for MAPE-K elements that serves as a basis for the behavioral aspects of HAFLoop. In the rest of this section, we provide the details of both concepts.

3.1. The hierarchical inter–intra collaborative architectural pattern (HIIC)

The hierarchical inter–intra collaborative architectural pattern, HIIC [17], has emerged by the need of a solution for supporting feedback loops with a varying degree of decentralization as well as loops able to interact with other loops. HIIC extends the notation described by Weyns et al. [5] for decentralized control in SASs, and combines and extends the hierarchical and collaborative patterns proposed by the same authors. HIIC consists of: a *bottom layer*, corresponding to the domain specific ME's logic; a *middle layer*, in charge of the adaptation of the domain specific MEs, placed on the bottom layer; and a *top layer*, for assessing middle layer loop operation (see Fig. 3). Moreover, in HIIC, MAPE-K elements can communicate with other MAPE-K elements of the same nature (e.g., a Monitor with other Monitors), and with one or more MAPE-K elements of other type (e.g., a Monitor with two Analyze elements). In order to do that, cardinalities have been introduced to the *Inter-components* and *Intra-components interactions* (see Fig. 3).

The decentralization degree of the elements of the loop can be freely changed thanks to the *MAPE-K component configurations* (MC, AC, PC, EC and KC in Fig. 3) and the *Configuration-component interactions*. The component configurations contain all the knowledge that a MAPE-K element needs to perform its functionalities (e.g., in the case of the Monitor, the monitoring frequency, the variables to monitor, etc.). Apart from enabling the decentralization of the loop, the explicit representation of elements' knowledge as a separate entity allows their later reconfiguration without affecting the operation of the rest of the elements of the loop. In this work, we apply the HIIC pattern to the HAFLoop architecture and propose the mechanisms required for enabling such reconfiguration at runtime (missing in the definition of the HIIC pattern). Moreover, in HAFLoop, the HIIC pattern is used for modeling the decentralized interaction of the elements as well as their communication with elements of other loops.

3.2. The FESAS component template

The FESAS component template [18] is part of the FESAS project framework [19], and has emerged as a solution for simplifying and fastening the development of MAPE elements through the reusability of components. The template describes an implementation-independent reusable MAPE element (see Fig. 4). A FESAS MAPE element is composed of an exchangeable logic (e.g., for the Analyze element, this would be an algorithm for analyzing monitoring data) and logics for communication and data handling. Moreover, it provides interfaces for receiving and sending data to other elements as well as requesting data from other elements. The division of communication and data handling functionalities in subcomponents, as well as customized functional logic, enables the reuse of subcomponents among the different MAPE elements as well as different MAPE-K loops.

In this work, we extend the FESAS component template with adaptation capabilities for facilitating the design and development of adaptive MAPE-K elements. Moreover, we define and provide detailed descriptions of element's components and subcomponents as well as the mechanisms required for coordinating their operation with their adaptation process. Finally, in our proposal, the Knowledge base element is modeled using the same extended template. These design decisions make our proposal a complete and reusable solution for developing adaptive MAPE-K loops.

4. Related work

One of the most common approaches for supporting adaptive AMs is to separate the logic in charge of MEs' adaptation from the logic in charge of the adaptation of the AM. That is, to adopt an external approach. Typically, three layers, as in the HIIC pattern [17], are considered: one for the ME, a second one for the AM and a third one for the logic in charge of AM's adaptation. Examples of solutions that adopt this approach are the 3LA approach [20], ActivFORMS [9,21,22], PLASMA [23,24], ALM [10,13], Service ensembles [25], recent extensions of MORPH [26,27],

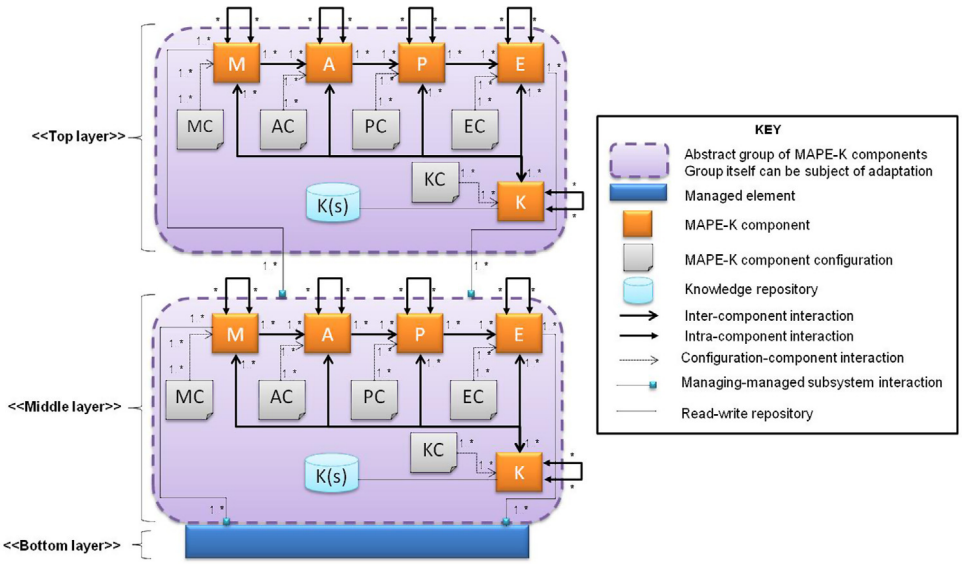


Fig. 3. Hierarchical inter–intra collaborative pattern (HIIC) [17].

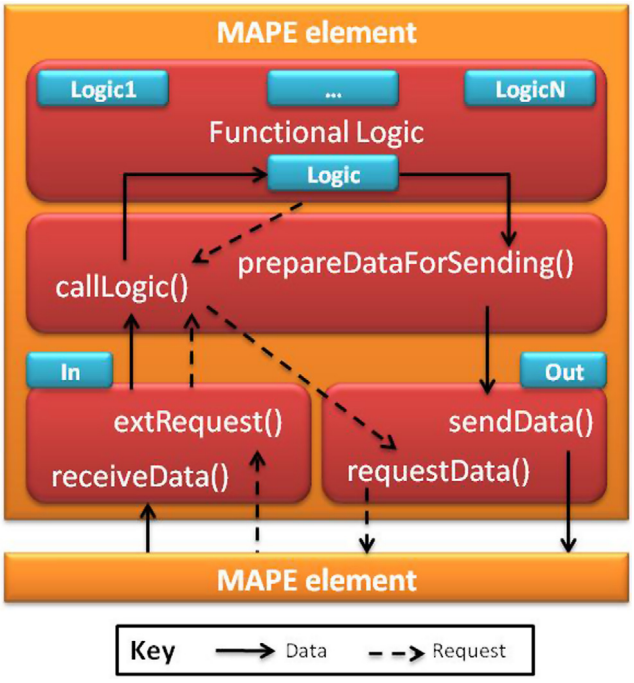


Fig. 4. FESAS component template [18].

among others. In order to support the third layer, most of these approaches consider the implementation of a second feedback loop that monitors the performance of the SAS AM and adjusts it, if necessary, at runtime. Although the idea of considering three layers is quite generic, current proposals for realizing it present limitations.

For instance, the approaches proposed by Gerostathopoulos et al. [28,29] as well as some of the approaches mentioned before such as 3LA [20] and PLASMA [23,24], focus on the adaptation of the adaptation rules/capabilities, managed by the AM for guiding the ME's adaptation process. Therefore, other types of adaptation are not supported, e.g., the structural adaptation of the MAPE-K elements of the AM which, as mentioned before, is necessary in a variety of situations. The three-layer solution called DAS [30] tries to address this issue by allowing different types of adaptation which include changes on ME's variability model, the techniques utilized for reasoning about ME's adaptation, as well as changes on a context model that contains the environmental variables considered during the ME's adaptation process. However, DAS [30] does not provide details for engineering of such vision.

In previous works, we have also proposed a three-layer approach. The former proposal, ACon [31], and its extension, SACRE [17,32], describe an approach for adjusting SASs' adaptation rules, the so called contextual requirements, through learning techniques (see Table 1). Learning-based solutions have also been proposed by other approaches such as OTC [33–37], NoMPRoL [38–40], KAMI [41], FUSION [42,43], DSPL [44–47], Adaptive KBs [48], and the approaches proposed by Zhao et al. [12], based on reinforcement learning, Rodrigues et al. [49], using data mining techniques, and the solution for mobile systems proposed by Pascual et al. [50]. The majority of these approaches focus on the ability of the learning techniques for discovering new adaptation rules or correcting and discarding existing ones. Therefore, other types of adaptation are not considered, and the engineering support for reusing such solutions is missing in most of them.

Moreover, learning-based approaches tend to see the adaptive behavior of AMs as an enhanced operation and not as a completely independent process. Therefore, some of the proposals rely on internal solutions, i.e., the MAPE-K elements are in charge of their own adaptation or the MAPE-K loop is extended by one or two extra elements. The adoption of internal solutions constrains the scalability of the proposals and may affect the performance of the AM regarding its main task, i.e., adapt the MEs [1]. This is also the case of the Auto-adjust [51] and the ESOs [8,52–54] approaches. For example in ESOs, the Monitor and Execute elements' implementations are selected by the MAPE-K loop itself at runtime.

There are also approaches that support the addition and removal of the complete AM at runtime. Examples of these approaches are DCL [57] and the approach proposed by Ali & Solis [61] for the self-adaptation of mobile resources. In these approaches, the AM is substituted by a new one when it is not suitable for the ME anymore, e.g., when its knowledge becomes obsolete. Finally, there are other approaches that even they are useful for solving relevant problems; their adoption for addressing other SASs' issues is limited. This is the case of Reqs@RT [58–

Table 1
Overview of works related to HAFLoop.

Approach	Trigger time	Type of change	Approach	(De)Centralization	System type
DYNAMICO [11,55]	Reactive	Structure	Rules	Centralized	Objective-driven
ESOs [8,52–54]	Reactive	Structure	Rules	Centralized	Component-based
RINGA [56]	Reactive	Parameter	Model checking	Centralized	Model-based
Gen. & evol. of adaptation rules [12]	Both	Parameter	Learning	Centralized	Goal-oriented
ACon & SACRE [17,31,32]	Both	Parameter	Learning	Both	Context-aware requirements
3LA [20]	Reactive	Undeclared	Generic architecture	Centralized	Goal-oriented and component-based
ActivFORMS [9,21,22]	Reactive	Parameter	Rules	Centralized	Goal-oriented
NoMPRoL [38–40]	Reactive	Parameter	Learning	Centralized	Component-based
DCL [57]	Undeclared	Structure	Replacement	Centralized	Goal-oriented and component-based
PLASMA [23,24]	Reactive	Parameter	Human-assisted	Centralized	Goal-oriented and model-based
FUSION [42,43]	Reactive	Parameter	Learning	Centralized	Feature-oriented and model based
KAMI [41]	Both	Parameter	Learning	Centralized	Model-based
OTC [33–37]	Both	Both	Learning	Both	Application-specific
ALM [10,13]	Both	Both	Generic architecture	Centralized	(Any)
DSPLs [44–47]	Both	Parameter	Learning	Centralized	Feature-oriented and model based
Reqs@RT [58–60]	Reactive	Parameter	Human-assisted	Centralized	Goal-oriented
Adaptive KBs [48]	Reactive	Parameter	Learning	Centralized	Goal-oriented and model-based
Service ensembles [25]	Reactive	Parameter	Rules	Centralized	Component-based
Auto-adjustment [51]	Reactive	Parameter	Rules	Centralized	(Any)
DAS [30]	Reactive	Both	Rules	Centralized	Model-based
MORPH [26,27]	Both	Both	Generic architecture	Centralized	Goal-oriented and model-based
Meta-adaptation [28]	Reactive	Parameter	Rule-based	Centralized	Model-based
Mobile resources [61]	Reactive	Structure	Replacement	Decentralized	Service-oriented
Self-adapt CVL [50]	Reactive	Structure	Learning	Centralized	Model-based
Assurances enhancement [49]	Reactive	Parameter	Learning	Centralized	Goal-oriented
Architectural homeostasis [29]	Reactive	Structure	Homeostasis	Both	Component-based
HAFLoop	Both	Both	Generic architecture	Both	(Any)

60], RINGA [56] and DYNAMICO [11,55]. For instance, DYNAMICO [11,55] proposes a 3-loop solution for supporting context-aware SASs. The loops are dedicated to adapt specific aspects that are relevant for this kind of SASs. Therefore, they cannot be utilized for satisfying other type of requirements.

Finally, most of the approaches mentioned above do not support adaptive AMs in decentralized settings. Thus, their application in modern SASs is limited. Table 1 provides an overview of the main characteristics of the approaches mentioned above. That is, the time at which an adaptation is triggered (proactive, before an event happens, or reactive, once it happens); the type of change enacted over the AM, parameter or structure; the fundamentals of the approach adopted, e.g., learning based, driven by predefined rules/functions or based on the replacement of the complete AM; the (de)centralization level supported; and finally, the type of system in which they can operate. These dimensions are based on the taxonomy for self-adaptation proposed in the extensive survey conducted by Krupitzer et al. [62]. For comparison purposes, we have also characterized our solution, HAFLoop.

Many efforts have been done for supporting adaptive AMs in SASs in order to improve ME's adaptation results. However, from the software engineering perspective, there are still some research gaps. A software solution should be reusable among different systems [63], in this case different SASs. From the analyzed approaches, we have concluded that there are not complete and flexible enough solutions that could be adopted by a variety of modern SASs. Therefore, there is a need and opportunity for engineers to innovate in this field. A reusable software solution would accelerate and structure the development process as well as facilitate the evolution of the system over time. Motivated by this facet, HAFLoop aims at providing a generic and reusable architecture for supporting engineers in the systematic development of adaptive feedback loops for SASs. Our proposal should be detailed enough to describe the operation of adaptive feedback loops' elements, components and subcomponents, and generic enough to be replicated in a variety of SASs.

5. HAFLoop

In this section, we provide the details of our proposal. First, we describe the architectural decisions that encompass the fundamentals of HAFLoop. Then, we explain how those decisions can be implemented as a generic framework for fastening the development of adaptive feedback loops.

5.1. Architecture

HAFLoop proposes a generic architecture for adaptive AMs. Concretely, since one of the most prominent loops for implementing AMs is the MAPE-K loop, our proposal bases its foundations in this loop. HAFLoop is able to support both structure and parameter adaptation of the elements of the loop, as well as operate in a variety of settings, from centralized to fully decentralized loops. The architecture consists of a set of modular components that can operate together or in isolation. Concretely, we have defined four types of reusable components that correspond to different abstraction levels, from more complex to simpler:

- Adaptive AM or adaptive feedback loop
- Adaptive MAPE-K element
- Element component
- Managers and policies

In next subsections, we describe each of these components.

5.1.1. Adaptive AM or adaptive feedback loop

In HAFLoop, we consider that the loop implementing the adaptive AM can have more than one element of each type, as we have established in the principles of the HIIC pattern [17] (see Section 3.1). This could be beneficial in some situations, e.g., for load-balancing, redundancy or for comparing two logics in a single SAS [30]. HAFLoop AMs can also share elements with each other in order to, for instance, unify adaptation decisions of different SASs. This is the case of complex SASs such as traffic control systems composed of a set of adaptive traffic lights, which operate as a system-of-systems [33–37]. Finally, by adopting the HIIC pattern, HAFLoop AMs' elements are able to communicate

and coordinate with MAPE-K elements of other SASs. For instance, in the case of our running example, an SDV could use vehicle-to-vehicle (V2V) communication for re-calculating routes to optimize the traffic in a city.

The structural and behavioral characteristics of HAFLoop AMs, mentioned above, are supported in by the use of runtime policies. Policies are configuration elements that, besides other information, encompass the up-to-date knowledge of AMs structure. These configuration elements have been previously introduced by our HIIC pattern [17]. However, in HIIC, details about how they should be managed are not provided. Therefore, in this work, we propose a series of mechanisms for managing and reconfiguring such policies at runtime, supporting in this way the adaptation of both the parameters and the structure of SASs' feedback loops. Policies also allow the loop elements to decouple their operation; therefore, fully decentralized loops can be supported. More details about policies are provided later in Sections 5.1.3 and 5.1.4.

5.1.2. Adaptive MAPE-K element

At the element level, HAFLoop separates the generic functionality, e.g., adaptation and data handling tasks, from the element-specific functionality, i.e., the logic required to monitor, analyze, plan, execute, and manage runtime knowledge. With this approach, a design for a generic adaptive element that serves as basis for all the elements can be provided. The HAFLoop MAPE-K element extends the FESAS template [18], presented in Section 3.2, with a set of components and mechanisms for coordinating elements' operation with their adaptation process. Concretely, a generic HAFLoop element is composed of four functional layers: a *Communication layer*, a *Message processing layer*, a *Logic layer*, and a *Knowledge layer* (see Fig. 5). These layers represent the main functionalities of an element, that is: communicate with other systems, process input and output messages, execute element-specific logic and adaptation logic, and manage runtime knowledge, respectively.

The *Communication* and *Message processing* layers correspond to the communication and data handling components in the FESAS template [18]. Meanwhile, the *Logic layer* partially corresponds to the logic component, since adaptation capabilities are not considered in the FESAS template. Finally, we have extended the original template with a *Knowledge layer*, in order to enable MAPE-K elements to manage element-specific runtime knowledge such as adaptive policies. In order to perform the functionalities of each layer, we propose the following components:

— *Communication layer*

- **Receiver**. This component is in charge of providing an interface for enabling external systems to communicate with the loop elements. It receives input messages and forwards them to the Logic selector component.
- **Sender**. The main function of this component is to send output messages to the corresponding recipients (e.g., other loop elements or the MEs) using the adequate interfaces, i.e., protocol, endpoints, etc.

— *Message processing layer*

- **Logic selector**. This component analyzes input messages and selects the logic component that should process them, i.e., the Functional or the Adaptation logic.
- **Message composer**. The Message composer is in charge of preparing elements' output messages. Output messages are mainly generated by the Functional and the Adaptation logic components. Concretely, the Message composer's function consists of determining the recipients of a specific message, ensuring format adequacy, and creating the necessary message copies. These copies are passed to the Sender component for being sent to the final recipients.

— *Logic layer*

- **Functional logic**. This component is the component in charge of enacting any logic related to the main functionality of the elements and is what gives them their nature, i.e., it determines whether an element is a Monitor, an Analyze, a Plan, an Execute or a Knowledge base.
- **Adaptation logic**. This component contains the logic for processing elements' adaptation messages, for instance, it could decide whether an adaptation action can actually be enacted or not, given a specific context. The Adaptation logic forwards adaptation messages to the Knowledge manager for being executed, and, if needed, sends output messages (e.g., an acknowledgment of the received adaptation message) to the Message composer.

— *Knowledge layer*

- **Knowledge manager.** The Knowledge manager, as its name implies, is in charge of managing the knowledge required by the rest of components for operating correctly. In our proposal, knowledge is stored in the form of runtime policies, which can be adapted at runtime. In order to support the adaptation process, this component receives adaptation messages from the Adaptation logic, then it determines to which element component(s) an adaptation request should be sent. This component can also be utilized for managing other types of knowledge. However, in this work, we focus only on the adaptive runtime policies since they play a crucial role in the adaption of the MAPE-K elements.

5.1.3. Element component

Elements' adaptations are managed at the component level. This decision makes our design modular and scalable since each element's component can be adapted independently from the rest of the components. In order to coordinate components' operation with their adaptation process, we propose to include in each component at least three subcomponents: a *Message manager*, a *Component policy manager*, and a *Component policy* (see Fig. 6). The *Message manager* subcomponent is dedicated to receive messages from other components (or external systems, e.g., in the case of the Receiver) while the *Component policy manager* receives the adaptation requests (from the Knowledge manager component).

After processing an adaptation request, the *Component policy manager* sends the corresponding adaptation action to the *Component policy* subcomponent. This last subcomponent represents the current active policy. After receiving the adaptation action, the *Component policy* performs two tasks: first, it updates the component's policy variables; second, it notifies the changes to the rest of the components' subcomponents that utilize such policy variables, e.g., the *Message manager* subcomponent described before. Fig. 7 illustrates this interaction.

Each subcomponent being notified, may then interpret such changes in the adequate way. For instance, in the case of the Analyze element's Functional logic subcomponents, a change on policy variables could mean the substitution of the algorithm used for analyzing monitoring data. In other cases, changes on policy variables may be interpreted as structural changes, for instance, in the example of the Analyze element's Functional logic, a structural change could be the substitution of one analysis provider by another, e.g., using re-composition techniques in case of service-based systems.

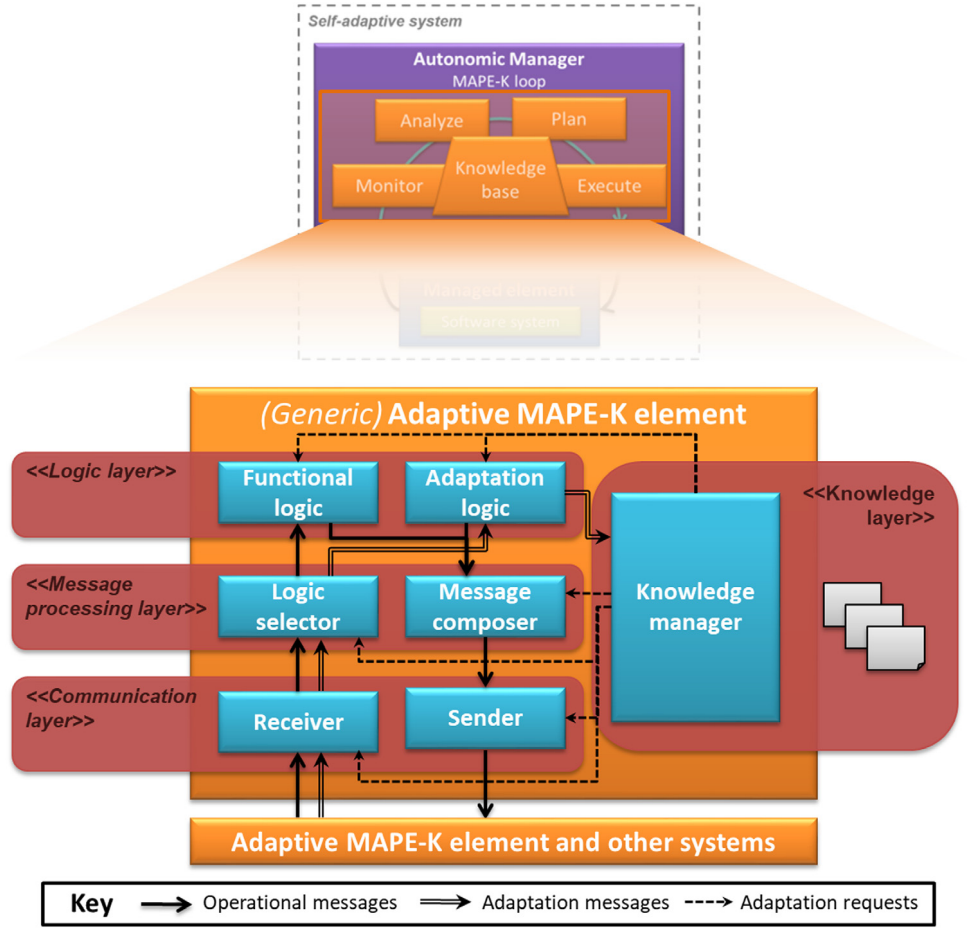


Fig. 5. Generic HAFLoop adaptive MAPE-K element.

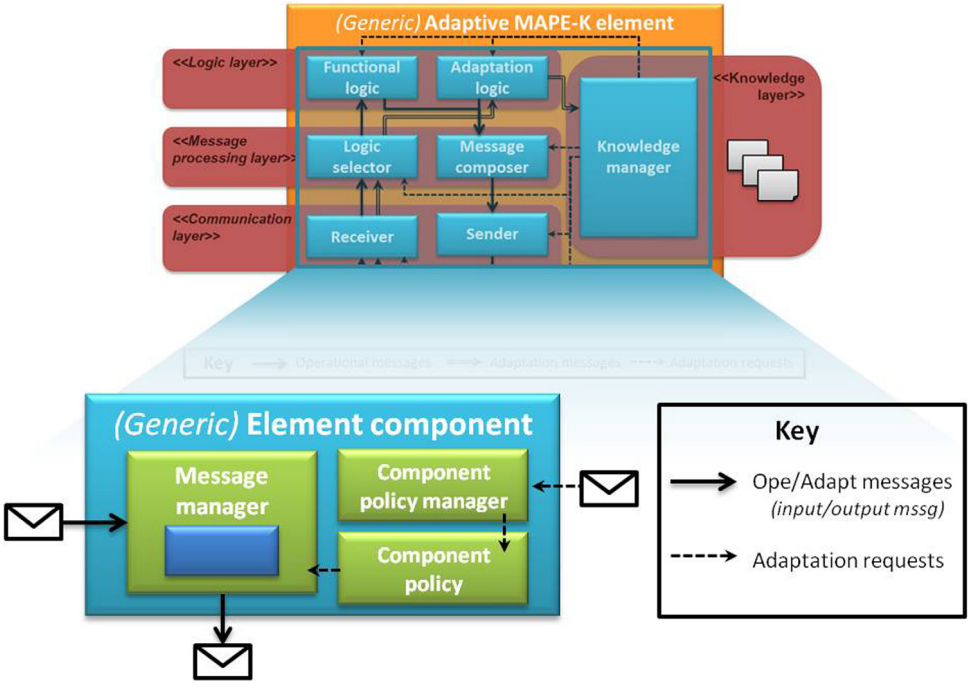


Fig. 6. Generic HAFLoop adaptive element component.

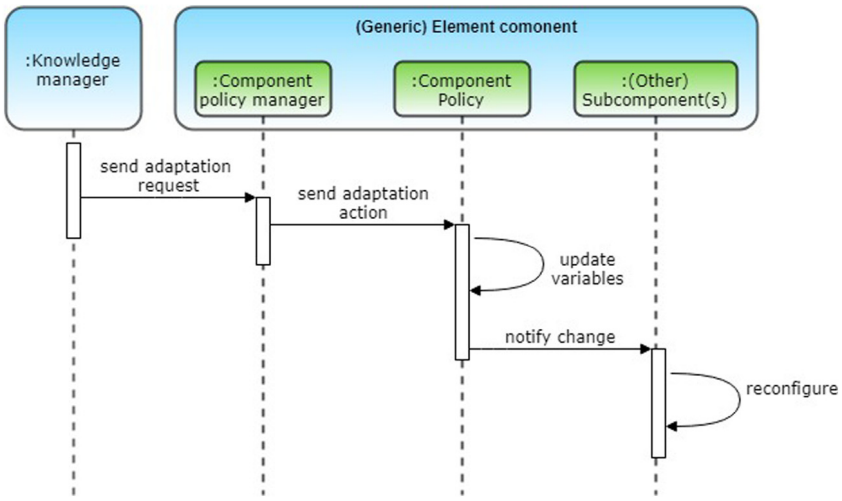


Fig. 7. Policy adaptation sequence diagram.

5.1.4. Managers and policies

Each component of an element has a different purpose, as described in Section 5.1.2; therefore, the component-specific logic encompassed by their *Message manager* subcomponent may differ for each of them. In order to support such variety, we propose a series of subcomponents for realizing the *Message manager* of each element's component (see Fig. 8). Below, we describe these subcomponents:

- **Message processor**. This subcomponent plays the role of the *Message manager* in the Receiver component. After receiving an input message, it utilizes Receiver's policy for deciding to which Logic selector a message should be sent and sends it.
- **Message dispatcher**. The Message dispatcher realizes Logic selector's *Message manager*. It utilizes component's policy for deciding whether a message should be sent to the Functional logic or to the Adaptation logic.
- **Functional logic enactor**. This subcomponent operates as the *Message manager* of the Functional logic component. Its functionality consists of calling the *Functional logic enactor manager*, another subcomponent of the Functional logic (*FLE manager* in Fig. 8), when a new operational message is received. The *FLE manager* in its turn decides to which specific logic a message should be sent.
- **Adaptation logic enactor**. This is the *Message manager* of the Adaptation logic component. It utilizes the Adaptation logic's policy for deciding whether an adaptation can be enacted in the element or not, in case of yes, the adaptation message is forwarded to the Knowledge manager component.
- **Formatter**. This subcomponent realizes the *Message manager* of the Message composer component. It utilizes the component's policy for determining: (1) to which recipients a message should be sent, (2) which data format is required by each recipient. Then, the Message creator, which is another subcomponent of the Message composer (see Fig. 8), receives formatted messages from the Formatter and sends the output messages to the corresponding Sender component.
- **Message sender**. This is the Sender's *Message manager*. Considering the component's policy, this subcomponent sends output messages to the elements' recipients using the corresponding interfaces, e.g., a service call.
- **Adaptive knowledge manager**. This subcomponent is the *Message manager* of the Knowledge manager component. It utilizes the component's policy for deciding in which component(s) of the HAFLoop element an adaptation should be enacted and sends the corresponding request(s).

HAFLoop components' operation is driven by adaptive runtime policies. The configuration variables contained in policies are intended to describe how an element should behave, how it is internally structured and how it communicates with other systems. Since policies' adaptation is already managed by HAFLoop components, as described in Section 5.1.3, SASs' owners can focus on designing how policy changes should be translated into changes of their specific software components.

Policies can contain innumerable configuration variables; variables will depend on the requirements of each HAFLoop instance, i.e., each use case. Moreover, variables can be generic and reusable among different SASs, but also domain specific. For instance, in our SDV example, policies could include the type of adaptation supported (structure, e.g., for enabling the trade-off of active sensors; parameter, e.g., for allowing changes on the monitoring frequency) as well as the type of logic that should be utilized for determining the need of adaptation (proactive, using learning techniques; reactive, once a sensor fault happens). In the evaluation of HAFLoop, we provide examples of policies (see Appendix); however, the list of variables considered is not intended to be complete, but to serve as guideline for future approaches reusing HAFLoop.

5.2. Implementation

In order to support engineers in the development process of adaptive feedback loops, we have implemented the generic functionalities of HAFLoop in the form of a Java-based framework. The framework consists of a series of interfaces that describe the behavior of the different HAFLoop components, i.e., the adaptive AM or adaptive feedback loop, the adaptive MAPE-K element, the element components and the subcomponents (i.e., managers and policies). The framework also provides a set of implementations for those interfaces, except for the Functional logic enactor manager (and the specific logics) and the Message sender, which will vary for each SAS. These implementations can be reused, substituted or extended by other SASs' owners. This framework is not only intended to serve as a basis for Java-based adaptive feedback loop projects, but also as an example implementation of HAFLoop for the future creation of frameworks for other programming

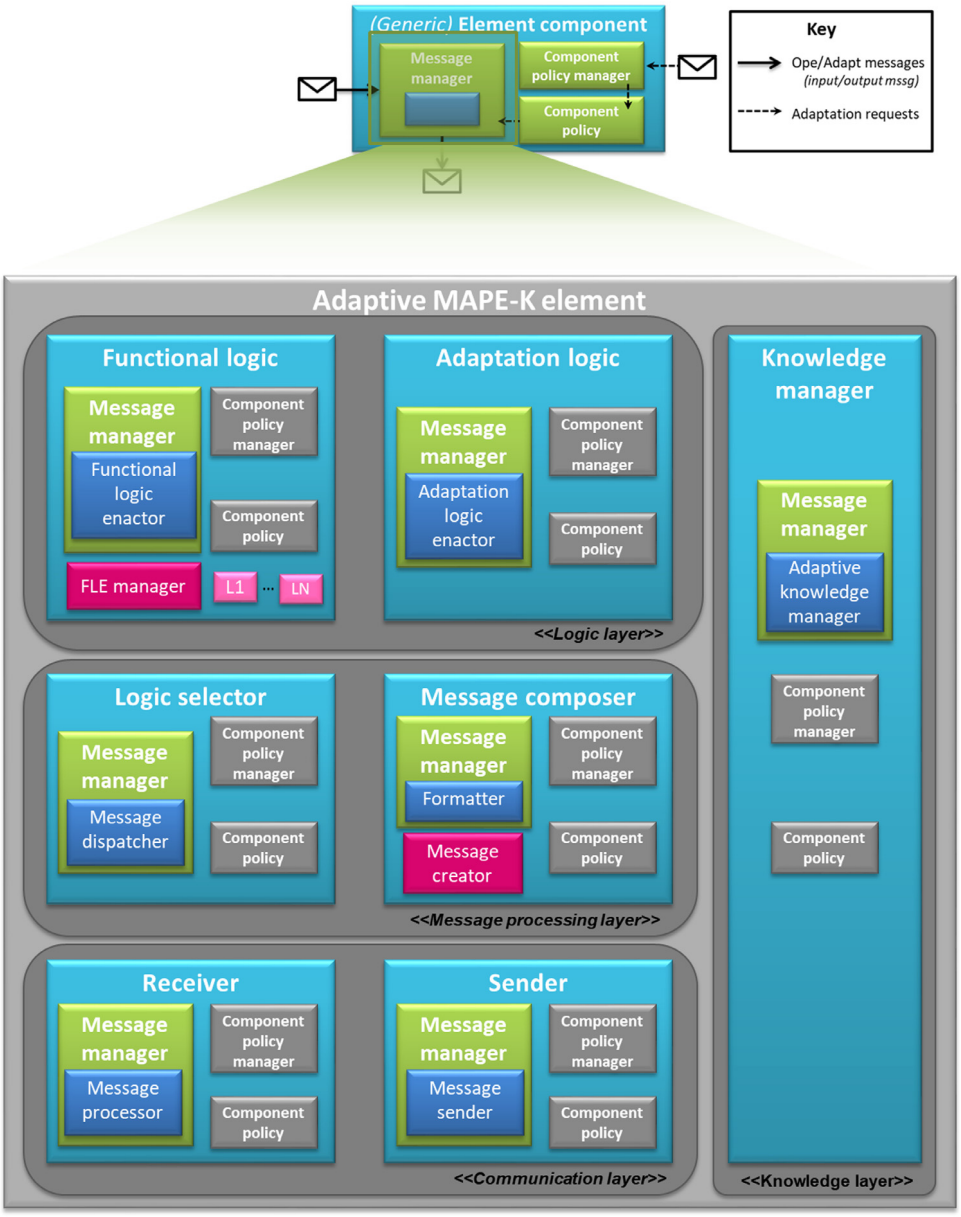


Fig. 8. Message manager realization in each HAFLoop element component.

languages. Fig. 9 shows a simplified version of the components of our HAFLoop implementation.

This implementation of HAFLoop improves the development process of adaptive feedback loops in different ways. First, the great majority of the components and subcomponents, as well as the communication mechanism, can be reused by any SAS. Therefore, systems' owners can focus on domain or application-specific issues, i.e., the development of MAPE-K elements' functional logics. Second, the operation of the components has been optimized based on previous experiences [17,32], utilizing popular software engineering techniques such as multithreading and asynchronous communications. Third, as mentioned above, components and subcomponents can be replaced by other implementations and/or extended for fulfilling specific SASs' requirements.

From an organizational perspective, since loop elements, components and subcomponents are conceptually and technically loosely coupled, they can be developed independently, e.g., by different specialized teams/companies, and gradually improved as required. This characteristic is quite convenient since nowadays software systems are developed more and more in distributed environments and following agile methodologies. Finally, regarding usability, due to the close relation between the terms typically used in the SASs' field and the HAFLoop components, we consider that our implementation is easy to understand, learn, and use. The source code of this implementation as well as more details about its construction and instructions for reusing it are open (under Apache License, Version 2.0) and available at https://github.com/edithzavala/loopa.

6. Evaluation

The evaluation of HAFLoop aims to demonstrate: first, the feasibility and benefits of adopting adaptive feedback loops in modern SASs; second, that our solution is generic enough to support SASs from different application domains and in different settings. With this purpose, we have considered a couple of SASs from two different domains: an SDV based on our running example, and an IoT network. On the one hand, as mentioned before,

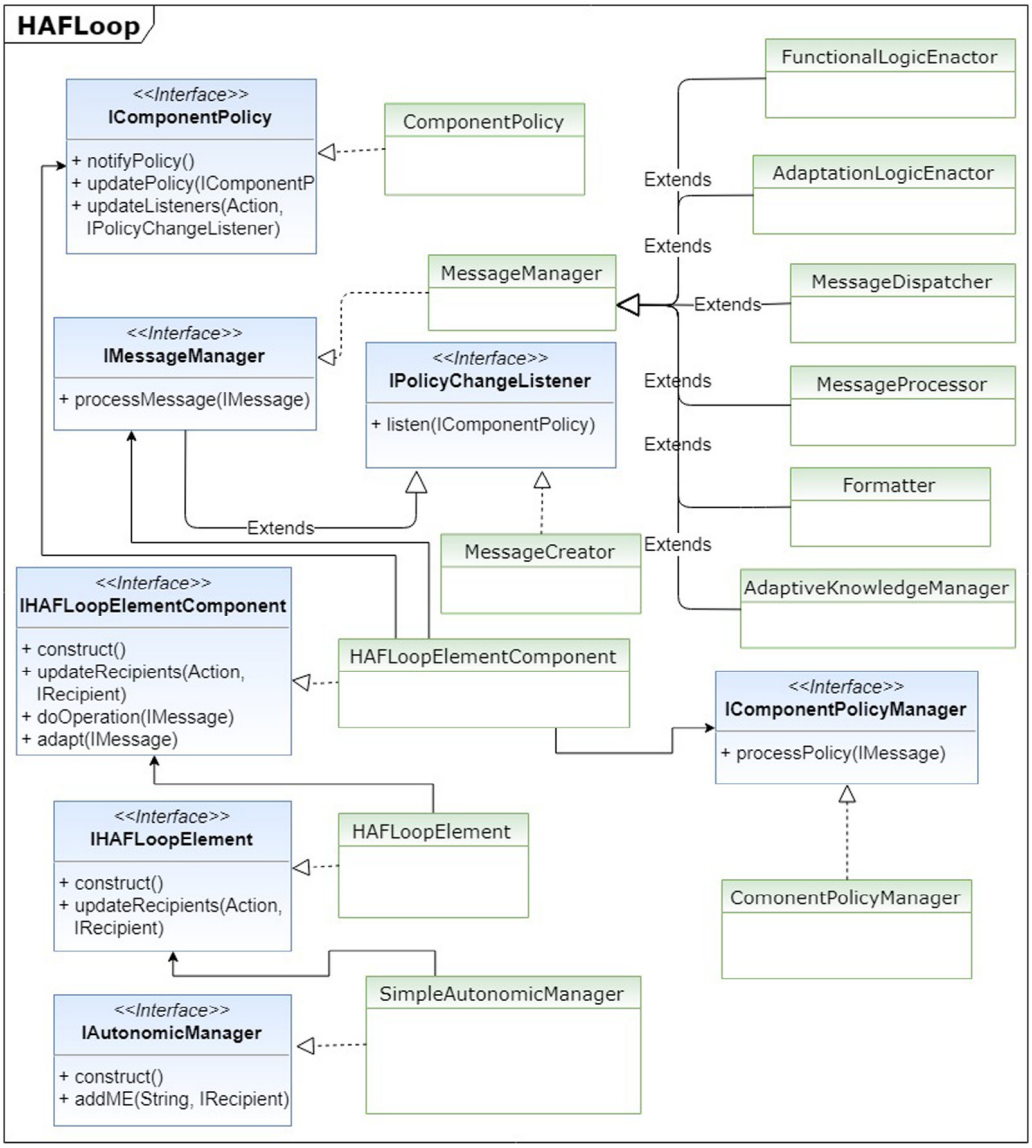


Fig. 9. HAFLoop framework implementation.

SDVs are currently a popular subject of research due to all the challenges their design, development and maintenance conveys. On the other hand, IoT applications are increasingly emerging and their rapid growth in recent years has brought multiple research opportunities (energy consumption optimization, availability, reliability, security, etc.) [64–66]. Moreover, both domains may share self-adaption solutions in the future due to its close relation, for instance, in the ecosystem of a smart city [67]. For both cases, we have simulated SASs through software components using existing platforms. The evaluation has been performed in real-time using an IntelR CoreTM i7-7700HQ CPU @ 2.80 GHz, with 16,0 GB RAM. On the one hand, the SDV has been simulated using the open-source software environment OpenDaVINCI [15]. On the other hand, the IoT network has been simulated using the DeltaIoT artifact [16]:

- OpenDaVINCI [15] is a middleware that provides the functionalities typically required for experimenting with autonomous vehicles, e.g., a visualization environment and components to embody simulations (vehicle kinematics, sensor simulations for a virtual camera, infrared, and ultrasonic sensors). Moreover, it provides a series of reusable algorithms for autonomous vehicles.
- Similarly, DeltaIoT [16] provides a simulator of an adaptive IoT network (composed of temperature, RFID and infrared sensors) for the evaluation of new solutions. In the simulator, the activities of the network during a specified period of wall clock time can be simulated in one run. As an example of usage, DeltaIoT [16] provides the logic of an adaptive IoT network (composed of 15 motes, 14 sensors and a gateway) that aims to minimize energy consumption.

For both SASs, we have reused the algorithms provided by the simulators for supporting the operation of the adaptation logic. In the remainder of this section, we describe the evaluation process and the threats to validity that we have identified for this evaluation.

6.1. HAFLoop instantiation

In order to conduct our evaluation, we have implemented adaptive feedback loops using the HAFLoop framework. Reusing all the generic modules of the framework, in both cases we only had to implement: elements' *Functional logic enactor manager*

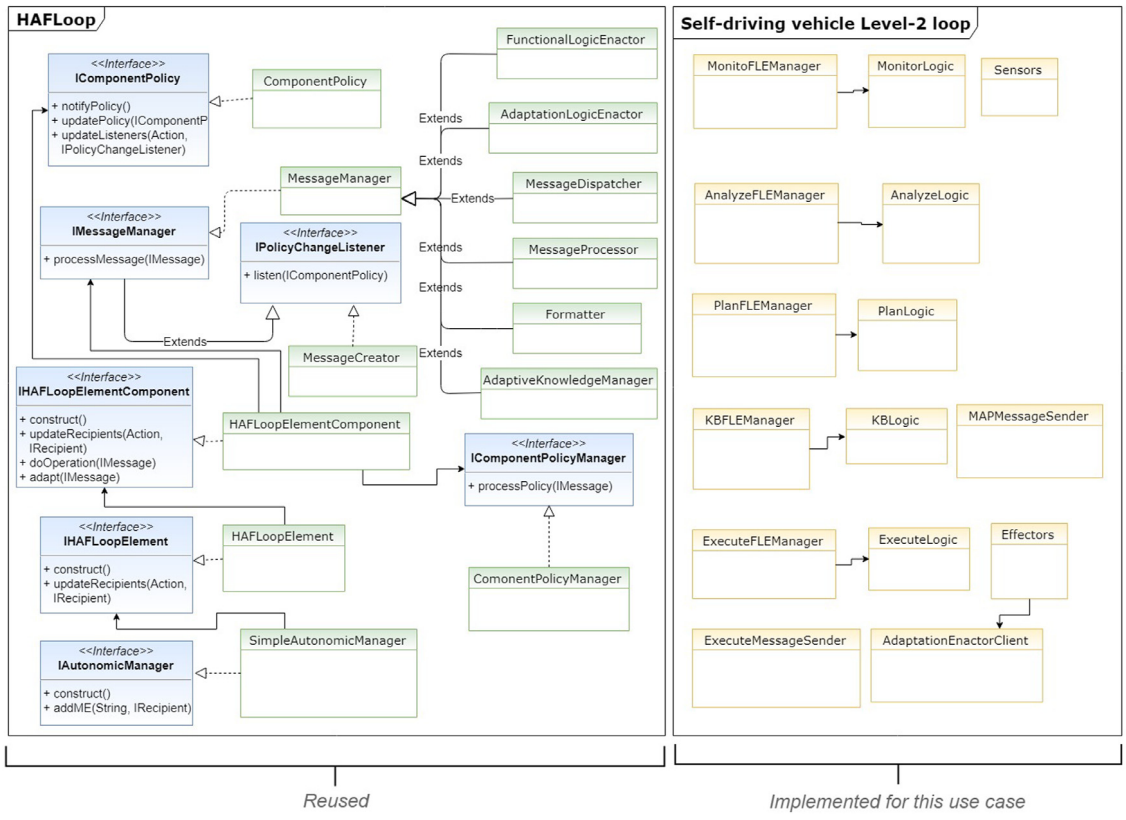


Fig. 10. Level-2 loop implementation using HAFLoop.

subcomponent (and the required use case logics), elements' Message sender subcomponent, and the *Sensors* and *Effectors* (for which our implementation of HAFLoop also provides interfaces). We have instantiated both centralized and decentralized loops; therefore, the flexibility of HAFLoop's architecture can also be demonstrated. Finally, in both cases, we have adopted the layer-based architecture proposed by the HIIC pattern. Concretely, we have implemented for each SAS two loops: a Level-1 loop (corresponding to the Middle-layer loop in Fig. 3) and a Level-2 loop (corresponding to the Top-layer loop in Fig. 3). These loops are completely decoupled, i.e., they operate as completely independent systems. Below, we describe the implementation of each SAS.

6.1.1. An improved context-aware self-driving vehicle

OpenDaVINCI provides the SDV logic in C++. This encompasses the vehicle itself and a feedback loop in charge of controlling the self-driving functionality, henceforth Level-1 loop. For this evaluation, we have extended such logic introducing runtime policy variables and context-aware functionality. Concretely, the Monitor element of the loop was extended by a weather service, a traffic service and V2Vcommunication. For the system in charge of the adaptation of the Level-1 loop, we have implemented a second loop, henceforth Level-2 loop. For implementing the Level-2 loop, we have used the HAFLoop framework. Fig. 10 provides a simplified overview of the implementation of this loop. Both loops are centralized and operate as independent services connected through an adaptor component. Concretely, each loop can be summarized as follows:

- Level-1 loop. This loop is implemented by a series of containerized services. First, a set of microservices implement the Functional logic of the Monitor element of the loop. Each microservice gathers data from a different source, being the sources: an inertial measurement unit (IMU), a camera, an infrared and ultrasonic sensors system, V2V communication, and weather and traffic services. Then, the Analyze, Plan and Execute elements' logics are implemented by another service that manages the self-driving context-aware functionality, taking into account the monitoring data gathered at runtime. All these services communicate with each other through a multicast service, placed at the Knowledge base element of the loop. The implementation of this loop can be found at https://github.com/edithzavala/OpenDaVINCI/tree/feature.smartvehicle and https://github.com/edithzavala/cityreporter.
- Level-2 loop. For this loop, we have reused the generic components of the HAFLoop framework and instantiated the *Simple Autonomic Manager* implementation. The AM consists of a loop composed of one element of each type (see Self-driving vehicle Level-2 loop in Fig. 10). The setup of this loop is carried out by the AM module, which receives all the policies for initializing and connecting elements and elements' components. It is also in charge of notifying the corresponding elements, when a new ME is connected. On the left side, the HAFLoop generic modules available for the implementation of the loop; on the right side, the modules implemented for this specific SAS. The complete implementation of this loop can be found at https://github.com/edithzavala/ksam-loopa.

6.1.2. An improved adaptive IoT network

DeltaIoT provides the adaption logic of the IoT network in a single Java class. This corresponds to the Level-1 loop of the adaptive IoT network. In order to make this loop adaptive as well, we have migrated such logic to a HAFLoop loop. Fig. 11 provides a simplified overview of the implementation of this loop (IoT Level-1 loop modules). In this case, the implemented loop

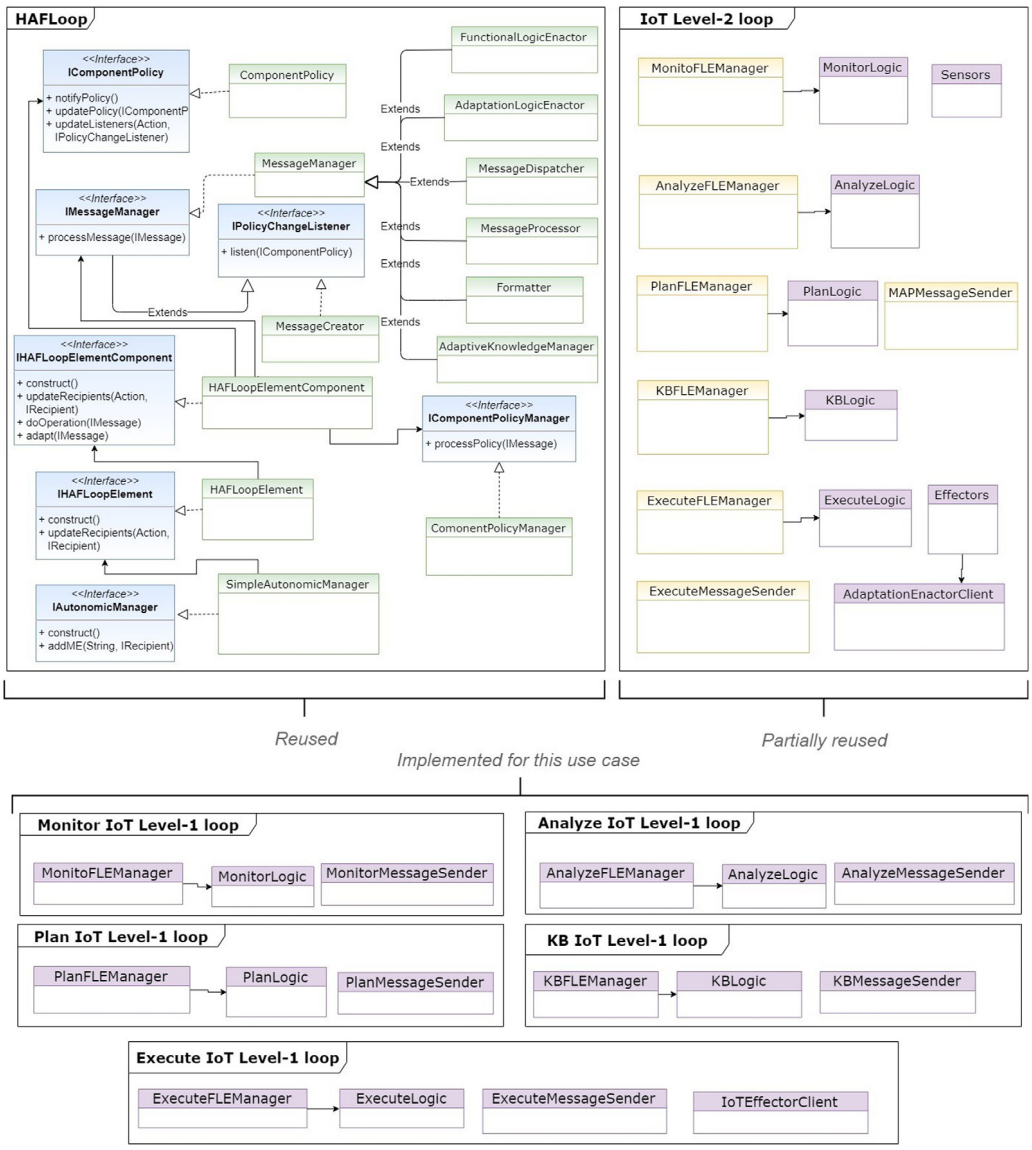


Fig. 11. Level 1 and Level-2 loops implementation using HAFLoop.

is decentralized. For adapting the Monitor element of the Level-1 loop, we have reused the structure and some of the logic and policy variables of the HAFLoop SDV Level-2 loop.

- Level-1 loop. This loop is composed of a series of services. Each element of the loop is an independent service and has been developed using the implementations of HAFLoop. This is possible thanks to the modularity offered by our approach, both at design and implementation levels. The structure of the Level-1 loop for the IoT network is dynamic. Concretely, the number of Monitor instances operating at the same time changes during the execution. The amount of monitors running at a specific point of time is transparent for the IoT network as well as for the rest of the elements of the Level-1 loop. *Effectors* and *Sensors* are implemented by the DeltaIoT simulator; therefore, in this case they are not included. Finally, setup and policies' management processes are carried out by each element, in a decentralized way. The complete implementation can be found at https://github.com/edithzavala/DecentralizedLoop-HAFLoop.
- Level-2 loop. This loop instantiates the *Simple Autonomic Manager* implementation provided by HAFLoop, which manages setup and policies as explained for the SDV case. Some changes on the elements' implementation have been done, e.g., on the functional logic (represented by the elements' Logic classes in Fig. 11), in order to correctly adapt the DeltaIoT Level-1 loop. Other components that have changed are the interfaces of the loop for interacting with the Level-1 loop. The rest of the components have been reused from the SDV use case. The complete implementation of this loop can be found at https://github.com/edithzavala/DeltaIoTLoopa2MAPEK.

6.2. Scenarios' execution

6.2.1. An improved context-aware self-driving vehicle

The evaluation of HAFLoop in the SDVs domain has consisted in two main use cases: a sensor fault and battery level issues (see Table 2).

In this evaluation, the adaptation decisions are based on three main factors: the number of vehicles on the road (the more vehicles, the more increased driving risk); the typical self-driving functionality usage, learned from driver's behavior (using data mining techniques) which has been simulated in a training phase;

Table 2
SDV use case scenarios.

Use case		Scenario	Expected adaptation
Sensor fault	us_1	Frontal ultrasonic sensor fails when the SDV goes on a road with no other vehicles, at the beginning of the journey.	No adaptation is enacted. Self-driving functionality stays active.
	us_2	Frontal ultrasonic sensor fails when the SDV goes on a road with other vehicles, at the beginning of the journey.	V2V communication is activated given the increased driving risk. Self-driving functionality stays active.
	us_3	Frontal ultrasonic sensor fails when the SDV goes on a road with no other vehicles, close to the end of the journey.	No monitor adaptation is enacted. According to patterns learned, driver will change to manual mode in the near future.
Battery issues	us_4	Critical battery level is experienced when the SDV starts its journey in a road with no other vehicles. Parameter adaptation is not supported.	A trade-off between required and non-required monitor is performed, resulting in the deactivation of the traffic and weather monitoring services.
	us_5	Rapid battery depletion is experienced when the SDV is in the middle of its journey in a road with no other vehicles. Parameter adaptation is supported.	A trade-off between required and non-required data sources, and their monitoring frequency is performed, resulting in the proactive adaptation of the traffic monitoring service frequency (i.e., it is reduced).
	us_6	Critical battery level is experienced when the SDV starts its journey in a road with other vehicles.	No monitor adaptation is enacted given the increased driving risk. However, a take-over request is sent to the driver. Driving mode is changed to manual.

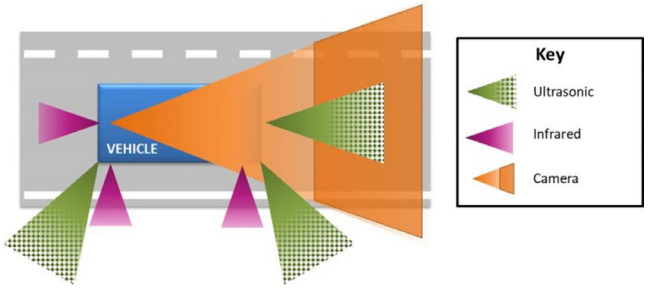


Fig. 12. SSDV sensors layout [15].

and the cost and utility of each source of monitoring data, i.e., sensors, V2V communication and cloud services. In this domain, we are interested in the adaptation response time given the criticality of this factor in this specific type of systems.

For running the evaluation of HAFLoop in the SDVs domain, we have utilized sensor data simulated by OpenDaVINCI, i.e., ultrasonic, infrared and camera. The sensors layout is displayed in Fig. 12. In order to find patterns on the self-driving functionality usage, we have utilized a set of data mining algorithms offered by a well-known data mining tool, Weka (https://www.cs.waikato.ac.nz/ml/weka/). In a previous work [17], we have already utilized the Weka tool in the domain of smart vehicles. The results in that work were satisfactory; therefore, we have incorporated the same tool in the evaluation of HAFLoop.

In order to train the SDV, we have considered the following scenario: a driver goes from work to her home in a daily basis and utilizes the self-driving functionality in specific segments of the journey. The resulting models are used at runtime for: (1) predicting the position of the vehicle in the near future (i.e., next N iterations) when a fault or battery issues are experienced; (2) predicting the self-driving functionality usage in that position. Given the nature of the data, for learning route preferences, we have utilized the IBk (K-nearest neighbors) classifier on vehicle's position data; meanwhile, for learning about the self-driving usage, we have utilized the JRip (Rule-based) classifier on a *Boolean* class variable that indicated whether the functionality was active or not. We have selected this algorithm based on previous works [17,31,68,69]. The resulting rules regarding the self-driving functionality usage are shown in Fig. 13.

At runtime, Level-1 and Level-2 loops require policy variables for driving their operation. Therefore, we have defined a set of policies for the loop elements. A simplified version of the most relevant configuration variables considered in the SDV evaluation, and the initial values assigned to them in each scenario, are provided in Appendix (Tables A.1 and A.2). For instance, some variables related to elements' structure such as the list of recipients, are not shown. Fig. 14 illustrates how each use case scenario has been executed, i.e., the number of vehicles on the road and the point (average point calculated after executing all scenarios' replications) at which the sensor fault or the battery issue is experienced.

When executing software systems, results may be affected by factors that are out of our control, e.g., the way in which the operative system manages resources in a specific execution. Therefore, in order to ensure the reliability of the results, we have decided to run each evaluation scenario several times. For calculating the correct number of replications to execute, we have used the formula of Berenson and Levine [70] and calculated the limit when the total population size (i.e., the expected number of executions in the software system life-time) tends to infinite:

$$n = \frac{Z_\alpha^2 Npq}{e^2 (N-1) + Z_\alpha^2 pq}$$

where:

- n is the number of replications needed (i.e., the sample size).
- Z_α is the value from the standard normal distribution for a selected confidence level. We have selected the typically used 95% of confidence level which Z_α is 1,96.
- N is the total population size. In our case, infinite.
- e is the sample error we accept for this evaluation. For the error, we have utilized 0.1.
- p and q: are the probability of success and failure respectively. We have selected the typical value of 0,5 for each of them.

As a result, we obtained n = 96,04. Based on this result, we have decided to run 100 replications for each use case scenario.

6.2.2. An improved adaptive IoT network

On the other hand, the evaluation of HAFLoop in the IoT network has been conducted for the following use cases: unexpected monitoring service shutdown and monitor QoS degradation. In this evaluation, a varying number of monitoring services are utilized for gathering data from the network motes, i.e., the

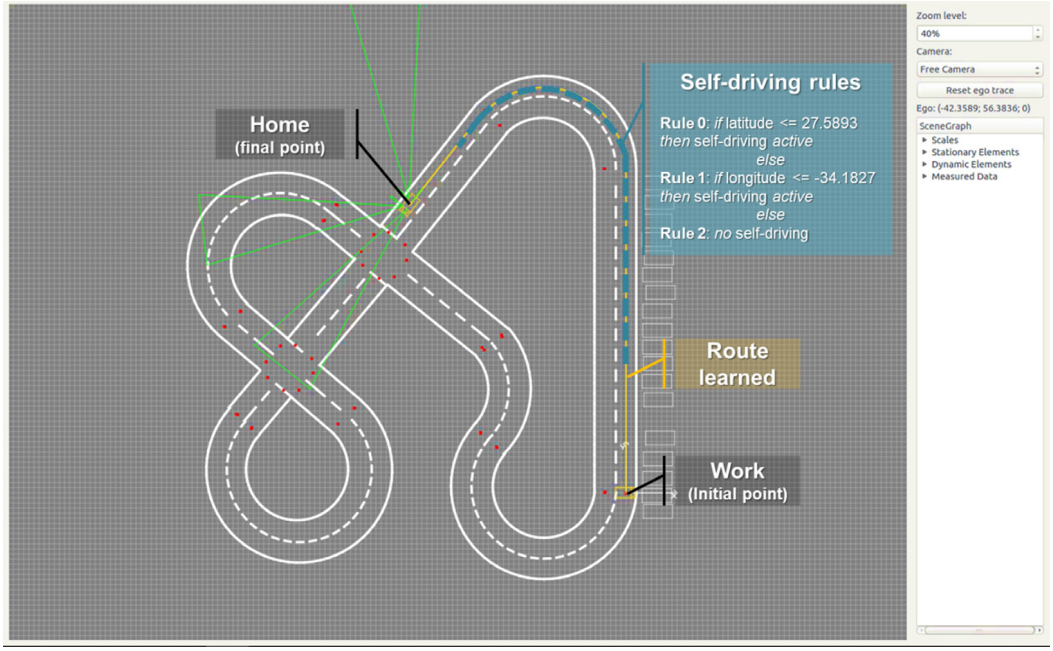


Fig. 13. Self-driving usage patterns in the simulation environment.

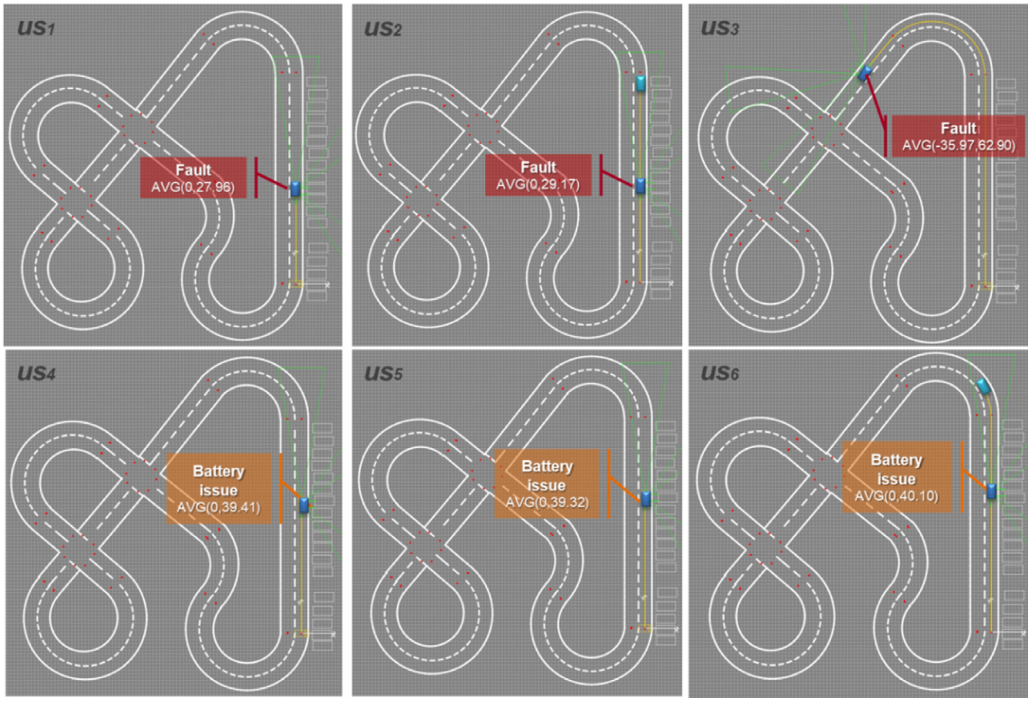


Fig. 14. HAFLoop evaluation context-aware SDV scenarios execution.

monitoring task is distributed, initially balanced among two monitoring services. The scenarios evaluated for each use case are shown in Table 3. The scenarios are designed to evaluate the benefits of enabling adaptive feedback loops against using static loops in adaptive IoT networks. For that purpose, we use the metrics relevant for this domain, proposed by Iftikhar et al. [16]: energy consumption and packet loss. The initial adaptive IoT network uses the Level-1 loop for maximizing network's life-time. That is, this loop is in charge of reducing energy consumption through the adaption of motes' operation. In this context, a malfunctioning loop would not be able to properly satisfy its goal.

In the DeltaIoT simulator, the activities of the network during a specified period of wall clock time can be simulated in one run; the default period is 15 min. The exemplar provides two predefined configurations: a default network configuration with 14 sensors and a gateway, as shown in Fig. 15; and, a reference configuration where each mote in the network communicates at maximum power, and sends/forwards all its messages to all its parents. Adaptation of the IoT network is based on adapting the network settings of the motes that participate in the IoT network. In our case, the configuration regarding the power used for transmitting messages is the one adapted at runtime.

Table 3
Adaptive IoT network use case scenarios.

Use case		Scenario	Expected adaptation
Monitor service shutdown	us_1	One of the monitoring services unexpectedly shuts down and Level-2 loop is not enabled.	No adaptation is enacted and the Level-1 loop is not able to operate correctly (e.g., the Analyze element does not have all the information it requires for determining the need of network's adaptation).
	us_2	One of the monitoring services unexpectedly shuts down and Level-2 loop is enabled.	A new Monitor microservice is started with the configuration of the lost Monitor service.
Monitor QoS degradation	us_3	One of the monitoring services experience performance degradation (response time) and Level-2 loop is not enabled.	No adaptation is enacted and the Level-1 loop is not able to operate correctly (e.g., at some iterations, monitoring data is incomplete and no adaptation is done).
	us_4	One of the monitoring services experience performance degradation (response time) and Level-2 loop is enabled.	A new Monitor microservice is started and a load-balancing process is triggered, resulting in the adaptation of the slow Monitor policy, .i.e. list of monitored motes is reduced.

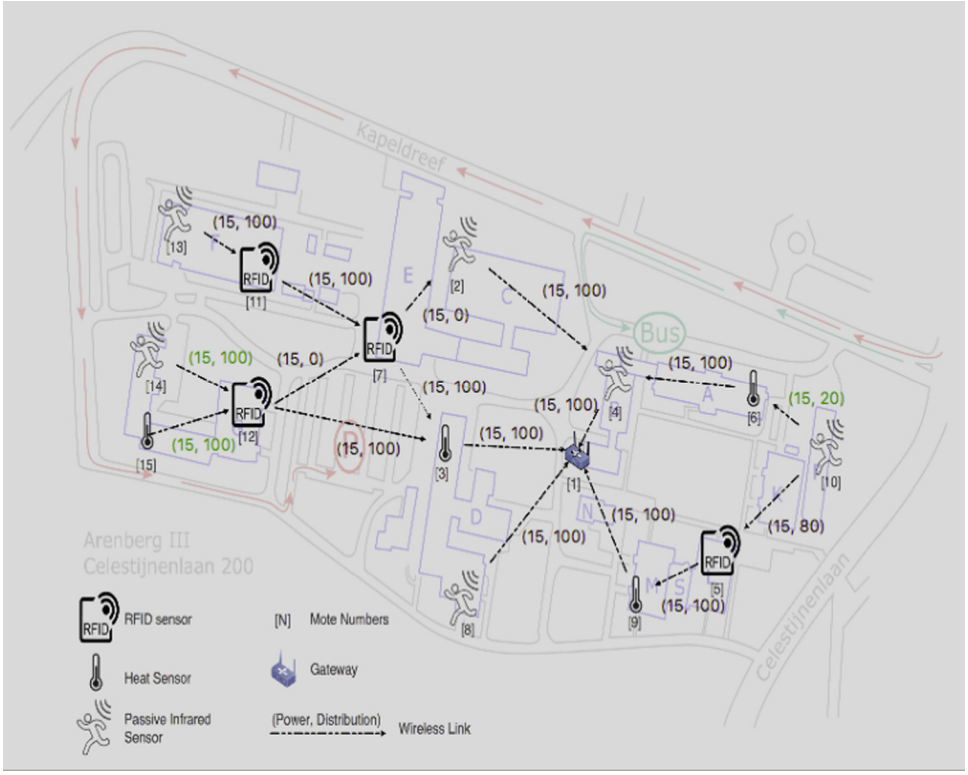


Fig. 15. DeltaIoT network topology [16].

In order to simulate traffic, the DeltaIoT simulator is loaded with different profiles. For the evaluation of HAFLoop, we have utilized a profile that simulates communication interference at some points of time as well as fluctuating traffic. This profile is the one also utilized by the authors for exemplifying an evaluation using DeltaIoT. Fig. 16 illustrates the execution of different DeltaIoT runs. Concretely, it shows the energy consumption and packet loss of a network over time for different scenarios. In both use cases of this evaluation of HAFLoop, monitor shutdown and monitor performance degradation have been simulated at the beginning of the execution, i.e., around the second or third iteration of the Level-1 loop.

In order to report results, we have calculated the energy consumption and packet loss average for a complete run. Then, we have computed the average of the results taking into account all the replications of a run. Finally, for this evaluation, we have defined a series of policy variables for the Level-1 and Level-2 loops. A simplified version of these variables, and the initial values assigned to them in each scenario, are provided in Appendix (Tables A.3 and A.4). Some variables, for instance, the ones related to the elements' structure, are not shown, but they are present in the implementation. In the list of policies, it can be noticed that some variables were reused from one application domain to another, although with different values. For example, the variable *Monitor*, which corresponds to the list of available monitoring sources, or the *Alert iterations* variable, which indicates the number of iterations that the Analyze element should wait before triggering an adaptation plan.

6.3. Analysis of the results

6.3.1. An improved context-aware self-driving vehicle

In order to analyze the evaluation results of the SDV, two aspects are explored: adaptation response time and adequacy. The response time is split into:

- Level-2 loop response time. Time elapsed since the Monitor element detects a sensor fault or a battery issue until an adaptation decision (in case of no adaptation required) or an adaptation request is sent to the Level-1 loop.
- Level-1 loop response time. Time elapsed since an adaptation request is received until it is enacted.

Fig. 16. HAFLoop adaptive IoT network scenarios example execution.

— Data mining response time. Time required by the data mining module for performing the predictions at runtime. This time is subsumed by the Level-2 loop response time but still we find interesting to isolate it in our benchmarking.

Regarding the adequacy, two metrics are taken into account:

— Adaptation enactment/decision correctness. Expected adaptations, described in Table 3, which are finally realized.
— Prediction correctness. The need for (no) adaptation is correctly predicted and prediction results are timely, i.e., SDV position after prediction is the same or previous to the last predicted.

Fig. 17 presents the Level-1 and Level-2 loops' response time values obtained in each use case scenario replication as well as the data mining module's response time. The *x*-axis of each sub-graph shows the number of replication (iteration), while the *y*-axis shows the response times. On the other hand, Table 4 provides the replications' average response time (in milliseconds) for each use case scenario. In Table 4, we also include the standard deviation of the response times. The Level-1 loop response time values range in average from 406 ms to 1557 ms. Regarding Level-2 loop response time, values range in average from 29,106 ms to 89,218 ms. Comparing these results with those obtained in a previous evaluation that we conducted in the domain of smart vehicles [17], a great improvement can be noticed. In our previous work, the response time of the scenarios using data mining was in the order of seconds while in this evaluation, results are in the order of milliseconds.

This improvement is due to different factors: (1) thanks to the utilization of our framework, software modules now communicate with each other asynchronously; (2) elements' operation and their adaptation process are treated independently, i.e., modules are multithread; (3) the amount of data to analyze at runtime has been drastically reduced while ensuring its relevance, as suggested in the conclusions of our previous work [17]; (4) data mining is used for prediction and not for model generation at runtime. On the other hand, the significant difference in Level-2 loop's response time between sensor fault and battery issues scenarios is due to a control mechanism that we have implemented to avoid adapting the system in response to isolated events.

In the case of a sensor fault, this mechanism forces the system to wait some iterations before the need of adaptation is triggered. The correct number of iterations to wait may depend on several factors. Therefore, it is still a variable subject of research for future implementations and that we have included as part of the Monitor and Analyze elements' functional logic policy (see Appendix). Finally, regarding the data mining module response time, it ranges in average from 113,30 ms to 142,68 ms. From the results, it can be noticed that in this evaluation, the performance of the data mining module has had a great impact on the Level-2 loop's response time. Therefore, future approaches adopting learning techniques for adapting SDV's feedback loops should address the optimization of this module.

Regarding adaptation adequacy, in all the scenarios, adaptation has been enacted when required and the decision of not adapting has also been correctly made (according to expected adaptations described in Table 2). Moreover, regarding the prediction correctness, both position and self-driving functionality usage have been predicted correctly in all the scenarios. As a final metric, we have determined the prediction timeliness. In order to do so, we have plot for each scenario:

— **Route**. The route followed by the SDV during the execution of the scenario.
— **Self-driving active**. The segment in which the self-driving functionality is usually active, according to patterns learned.

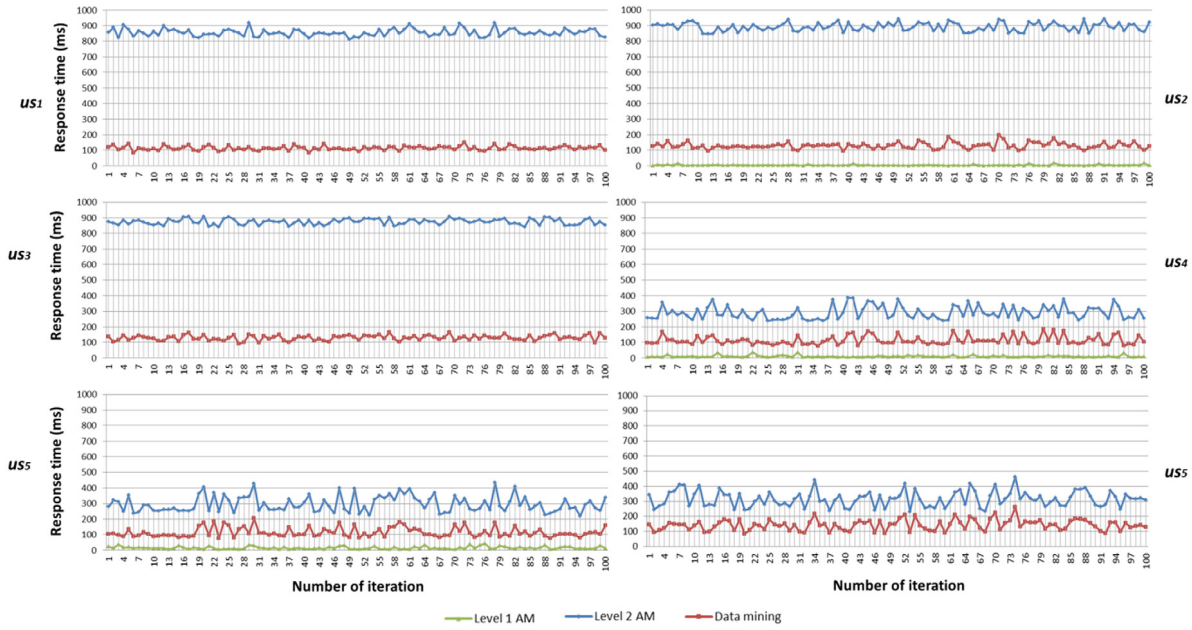


Fig. 17. Adaptation response time per use case scenario replication.

Table 4
Results per SDV use case scenario (time given in milliseconds).

Id	Level-1 loop response time	Level-1 loop response time std. deviation	Level-2 loop response time	Level-2 loop response time std. deviation	Data mining response time	Data mining response time std. deviation
us_1	N/A	N/A	855,34	23,34	113,93	13,31
us_2	4,06	3,56	892,18	27,16	129,68	19,44
us_3	N/A	N/A	875,35	18,35	130,47	16,39
us_4	10,82	6,36	291,06	42,39	113,30	27,45
us_5	15,57	8,62	297,00	51,50	115,03	30,45
us_6	N/A	N/A	315,50	52,34	142,68	35,12

— **Sensor fault/Battery issue position**. The average SDV position, taking into account the results of all the replications, at which the sensor fault or the battery issue is experienced.
— **Position after analysis**. The position of the SDV after executing the data mining and providing the predictions.
— **Last predicted position**. The last point predicted by the data mining algorithm.

The visualization of these variables is provided in Fig. 18. According to the results, we have concluded that in all the scenarios, the data mining predictions, apart from correct, have been timely performed. The number of positions to predict is indicated to the Analyze element through policies. Similar to the case of the waiting iterations mechanism, this is an exploratory variable that should be studied for each use case. The advantage of adopting HAFLoop is that SASs' owners can focus on investigating these application-specific variables instead of expending time on repetitive tasks such as the implementation of the generic functionalities of an adaptive MAPE-K loop.

6.3.2. An improved adaptive IoT network

In this section, we present the detailed results of the adaptive IoT network evaluation. In order to analyze these results, we have considered two aspects: adaptation impact and adaptation correctness. For the first metric, impact, two measures were considered:

— Average network energy consumption. The average energy consumed by the network during a run. Each run simulates a 24 h period in which every 15 min, motes' metrics are consulted.
— Average packet loss. Similarly, this corresponds to the average network packet loss rate of a run.

Table 5
Results per adaptive IoT network use case scenario.

Id	Average energy consumption (C)	Average energy consumption std. deviation	Average packet loss (%)	Average packet loss std. deviation
us_1	17,24	0,25	14,44	0,71
us_2	32,61	0,24	5,64	0,44
us_3	17,56	0,22	14,10	0,58
us_4	18,69	0,35	13,63	0,67
No network adaptation	33,03	0,43	5,39	0,94
Level-1 loop without faults	16,66	0,25	14,78	0,74

Regarding adaptation correctness, we were interested in assessing whether the expected adaptations, described in Table 3 are finally realized or not. In Table 5, we provide the replications' average energy consumption (in Coulombs) for each use case scenario. The standard deviation of the average energy consumption is also included. In Fig. 19 detailed average network energy consumption values obtained in each of the replications of each scenario can be visualized. In a similar way, Table 5 provides the information about the average packet loss and Fig. 20 the values

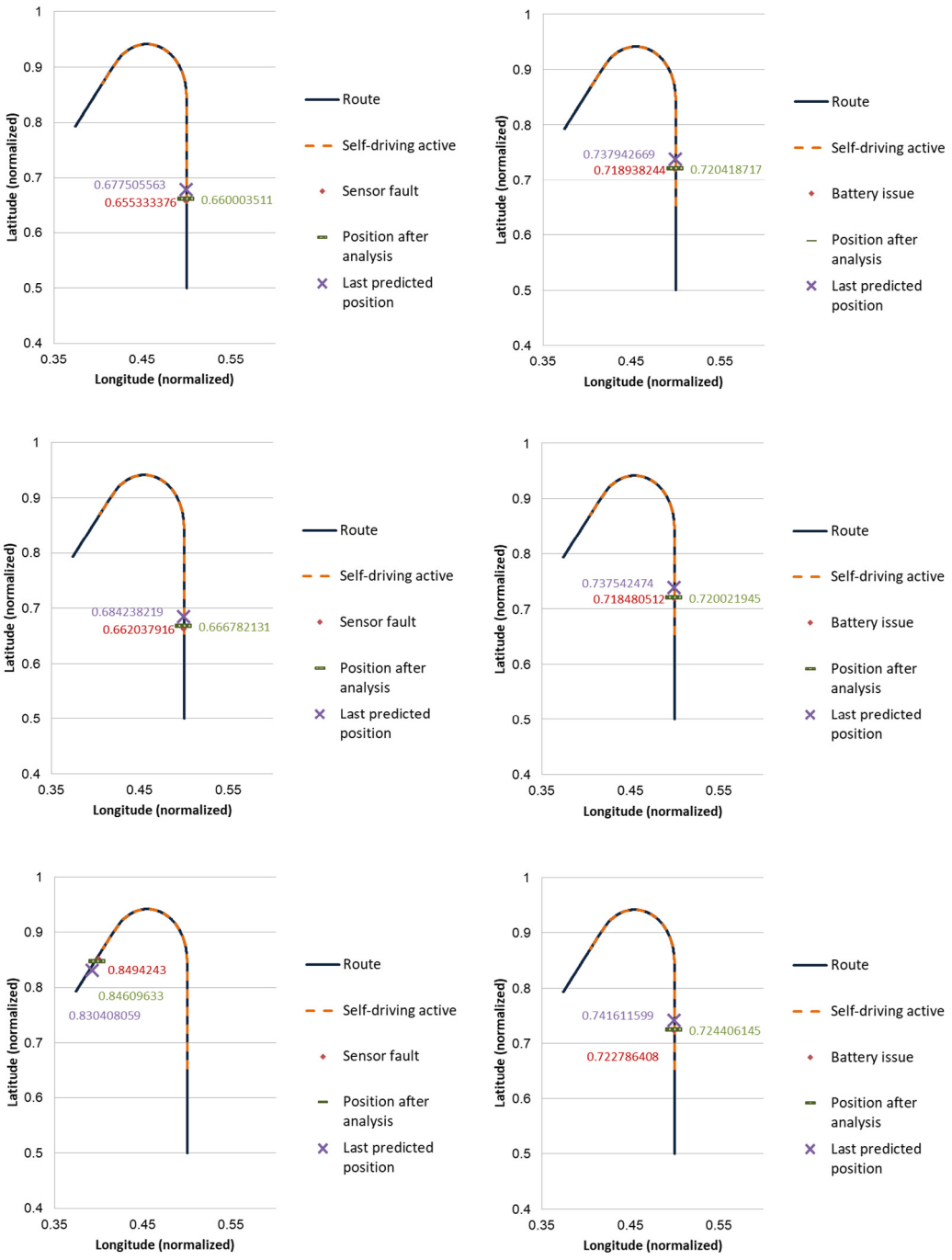


Fig. 18. Prediction timeliness per use case scenario.

obtained in each replication. Packet loss and energy consumption have a negative co-relation (see Table 5), because in order to reduce energy consumption, motes use less power to send packets. In this evaluation, we reused the algorithm provided by Iftikhar et al. [16], which focus on minimizing energy consumption while trying to reduce packet loss.

We have included in Table 5 the energy consumption and packet loss information for the cases when adaptation of the IoT network is no enabled (i.e., Level-1 loop is not operating), and when it is enabled and no problem is experienced (i.e., monitoring services operate without faults). These two scenarios allow us to compare our results against the worst and best cases. From the results shown in Table 5, it can be noticed that adopting adaptive feedback loops represents a great advantage for achieving SASs goals, particularly, when Level-1 loops are affected by runtime challenging situations such as faults.

The advantage of adaptive loops is more evident in the case of an unexpected monitor shutdown; see in Table 5 the resulting energy consumption for us_1 (it is clearly close to the energy consumption of a Level-1 loop without faults) compared to the resulting consumption for us_2 (which is closer to a non-adaptive IoT network). The results also confirm the importance of ensuring the adequate operation of SASs' feedback loops since any disturbance may negatively affect the adaptation process and in consequence the performance of the MEs. Regarding packet loss, results are consistent with the best and worst cases, i.e., more energy consumption means less packet loss and vice versa.

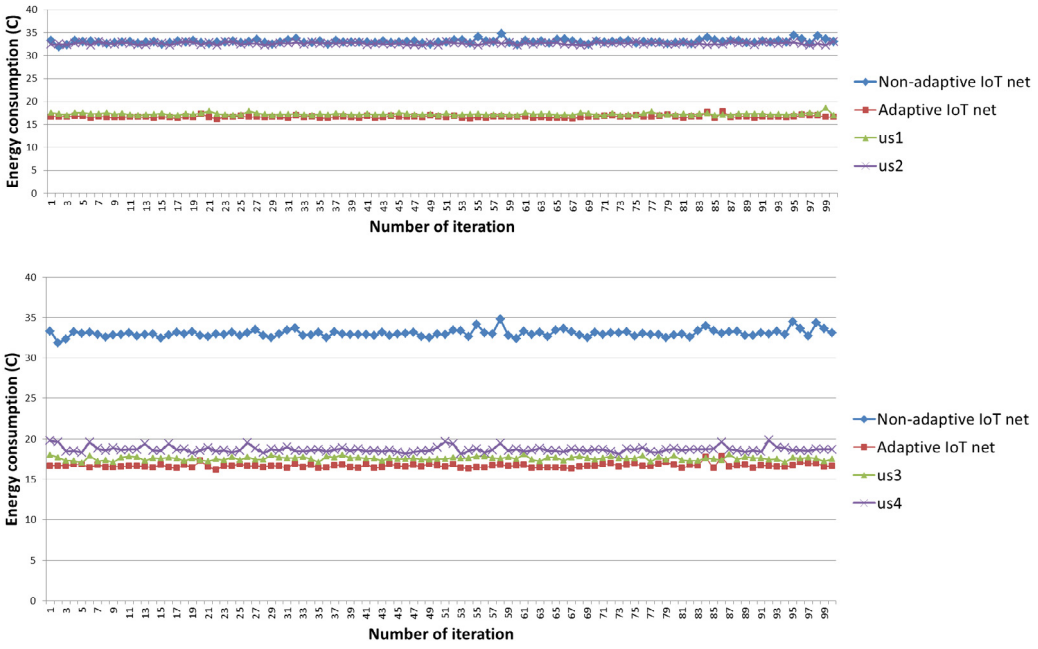


Fig. 19. Average energy consumption per use case scenario replication.

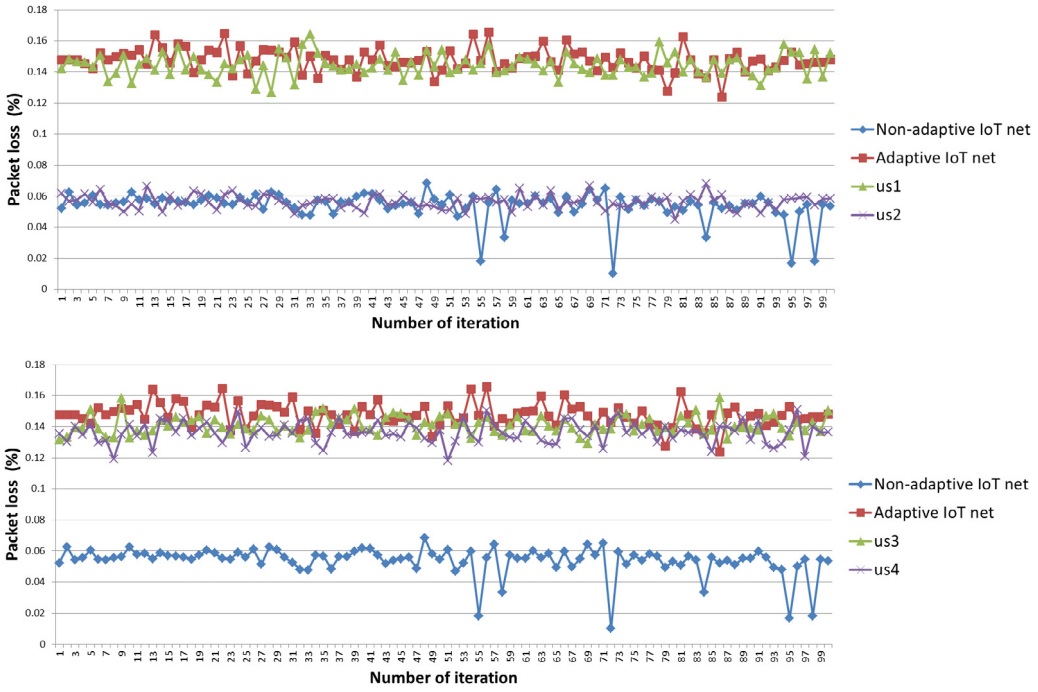


Fig. 20. Average packet loss per use case scenario replication.

6.4. Threats to validity

In this section, we describe the threats that affect the validity of this evaluation as well as the actions that we have taken for mitigating them:

- **Construct validity**. In this evaluation, a threat to construct validity is that it was conducted using simulated components. Thus, the evaluation could be affected by our interpretation of the environment and, in the case of the SDV, the interactions of the driver with the vehicle. Moreover, factors that can only be measured in a real environment, e.g., time required by a sensor for physically turning on and off, could not be reflected in our evaluation results. In order to reduce this threat, we have utilized the OpenDaVINCI middleware and the DeltaIoT simulator, which offer environments based on realistic data. The advantage of both platforms is that they provide the mechanisms required for transferring experiments setups to real-scale systems.
- **Internal validity**. The internal validity refers to our ability to reason about the results in each use case scenario, for instance, confounding variables' relationships. In order to reduce this threat, we have quantitatively interpreted our results using descriptive statistics to determine tendencies and dispersion. Accidental bugs in software components

Table A.1
SDV Level-2 loop policies.

Policy element	Variable	Variable description	us_1	us_2	us_3	us_4	us_5	us_6
Monitor	Alert iterations	Number of iterations, detecting sensor fault or battery issue, to wait before triggering an analysis alert		3			0	
	Initial battery level	The battery level at the initial point of each scenario execution		100%			60%	
	Battery limit	The battery level considered as critical				45%		
	Monitors	List of monitoring data sources, type of source (T), the monitoring data provided (Vars), the monitoring frequency (F) and their cost (C, a factor in relation to the rest of monitors, taking into account power and monetary aspects, and its utility for correctly supporting the self-driving functionality)	**traffic**: (T) service, (Vars) traffic factor, (F) 60.000 ms, (C) 40 **weather**: (T) service, (Vars) weather, (F) 60.000 ms, (C) 16			**traffic**: (T) service, (Vars) traffic factor, (F) 10.000 ms, (C) 40 **weather**: (T) service, (Vars) weather, (F) 10.000 ms, (C) 16		
			IMU: (T) sensor, (Vars) longitude–latitude- speed, (F) 100 ms, (C) 1 **camera**: (T) sensor, (Vars) image size–frontal distance, (F) 100 ms, (C) 10 **infrared (frontal right, rear and rear right)**: (T) sensors, (Vars) frontal right distance–rear distance–rear right distance, (F) 100 ms, (C) 5 each **ultrasonic (frontal center, front right, rear right)**: (T) sensors, (Vars) frontal center distance–frontal right distance–rear right distance, (F) 100 ms, (C) 3 each **V2V**: (T) service, (Vars) distance–road event, (F) 100 ms, (C) 30					
	Monitoring variables	Variables to be monitored and variables' values characteristics (type (T), min, max or possible values (Val)). Values out of min–max range or not listed as possible values, might indicate a monitor fault	**traffic factor**: (T) Double, (Val) −1, 10 **weather**: (T) String, (Val) Rain, Snow, Extreme, Clear, Clouds, Foggy, Fog, Drizzle, Mist **longitude**: (T) Double, (Val) −180, 180 **latitude**: (T) Double, (Val) −90, 90 **speed**: (T) Double, (Val) 0, 2 [m/s] **frontal right distance, rear right distance, frontal center distance, rear distance**: (T) Double, (Val) −1, 39 **road event**: (T) String, (Val) Crash **image size**: (T) Double, (Val) 0, 5.000.000					
	Initial monitors	Initial set of active monitoring data sources	Traffic, weather, IMU, camera, infrared (frontal right, rear and rear right), ultrasonic (frontal center, front right, rear right)					
Analyze	Alert iterations	Number of iterations, receiving an alert from the Monitor, to wait before triggering data mining analysis		3			0	
	Adaptation supported	Type(s) of adaptation supported	Structural		S		Param.	S
	Analysis technique	This could include: technique, tool, endpoint, algorithms, algorithms' parameters	**Technique**: Machine Learning **Tool**: Weka **Endpoint**: protocol, host, port **Algorithms**: JRip, IBk **Parameters**: Positions to predict (N = 100)					
	ME functionalities	Critical functionalities to be provided by the ME	Self-driving					
Plan	Plan technique	This could include: technique, tool, endpoint, algorithms, algorithms' parameters	Technique: Objective function (min (*monitoring cost*), max (*monitoring data required by the ME functionalities*))					
	Monitoring data required by ME functionalities	Monitoring data required by each of the ME functionalities	Self-driving: longitude, latitude, speed, frontal right distance, rear right distance and frontal center distance					
Execute	Level-1 loop endpoint	ME interface to communicate adaptation decisions	Protocol, host, port					
Knowledge base	Persistence format	The format in which data is going to be persisted	.arff (the format required by the Weka tool)					
	Monitoring data	List of monitoring data sources to take into account for persistence	Traffic, weather, IMU, camera, infrared (frontal right, rear and rear right), ultrasonic (frontal center, front right, rear right), V2V					

are also a threat to internal validity. We have tried to reduce this unavoidable threat using well-established frameworks and tools for building our solution such as Spring boot (https://spring.io/projects/spring-boot), Gradle (https://gradle.org/), Docker (https://www.docker.com/), among others.

– **External validity**. External validity refers to the generalizability of our conclusions. In this work, we have evaluated HAFLoop in two application domains. The results have shown the feasibility and benefits of using HAFLoop in modern SASs, particularly, SDVs and IoT networks. However, due to the simulation environments in which the scenarios have been executed, generalization may be limited, not only to these specific domains, but also to the application of HAFLoop in these specific domains. In order to reduce this threat, we have evaluated HAFLoop using different use cases. In future work, we plan to conduct a series of experiments using real vehicles in a scaled environment and the deployed IoT network of the KU Leuven.

Table A.2
SDV Level-1 loop policies.

Policy element	Variable	Variable description	us_1	us_2	us_3	us_4	us_5	us_6
Monitor	Monitors	List of monitoring data sources and their monitoring frequency						
	Monitoring variables	Variables to be monitored	Same as in Table A.1					
	Initial monitors	Initial set of active monitoring data sources						
Analyze	Vehicle variables	Variables resulting from vehicle's monitoring	Vehicle's position and speed, frontal right, rear right, frontal center and rear distance					
	Context variables	Variables gathered from external systems	Weather, traffic, road event					
Plan	Adaptation variables	Variables to include in the adaptation plan	Acceleration, steering wheel angle, driving route					
Execute	ME endpoint	ME interface to communicate adaptation decisions	Vehicle id					
Knowledge base	Level 2 AM sensors endpoint	Interface to send runtime data to Level 2 AM	Protocol, host, port					

Table A.3
IoT Level-2 loop policies.

Policy element	Variable	Variable description	us_1	us_2	us_3	us_4	IoT net	Adaptive IoT net
Monitor	Monitors	List of monitoring data sources, the monitoring data provided (MotesId) and the monitoring frequency (F)			**monitor1**: (MotesId) [2,3,4,5,6,7,8], (F) 2.000 ms **monitor2**: (MotesId) [9,10,11,12,13,14,15], (F) 2.000 ms			
	Initial monitors	Initial set of active monitoring data sources			Monitor1, monitor2			
	Pull frequency	Frequency at which Level-2 loop gathers data from Level-1 loop monitors				4.000 ms		
Analyze	Alert iterations	Number of iterations, detecting monitor shutdown or degradation to wait before triggering a plan alert	3	N/A	3	N/A		
Plan	Monitors	List of monitoring data sources, the monitoring data provided (MotesId) and the monitoring frequency (F)			**monitor1**: (MotesId) [2,3,4,5,6,7,8], (F) 2.000 ms **monitor2**: (MotesId) [9,10,11,12,13,14,15], (F) 2.000 ms			
Execute	Level-1 loop endpoint	Monitors interface to communicate adaptation decisions			Protocol, host, port			
Knowledge base	For this use case, we have not stored any data. All required knowledge was kept in memory.							

Table A.4
IoT Level-1 loop policies.

Policy element	Variable	Variable description	us_1	us_2	us_3	us_4	IoT net	Adaptive IoT net
Monitor	Monitoring variables	Motes to be monitored	Same as in Table A.3					
	Monitoring frequency	Frequency at which motes should be monitored						
	Analyze endpoint	Analyze interface	Protocol, host, port					
Analyze and Plan	IoT network composition	Total list of motes	**motes** = [2,3,4,5,6,7,8,9,10,11,12,13,14,15]					
	Plan/Execute endpoint	Plan/Execute interface	Protocol, host, port					
Execute	ME endpoint	IoT network interface to communicate adaptations	Protocol, host, port					
Knowledge base	For this use case, we have not stored any data. All required knowledge was kept in memory.							

7. Conclusions and future work

In this work, we have presented HAFLoop, an architectural proposal for supporting adaptive feedback loops in SASs. Although great efforts have been done for supporting the adaptation of SASs' AM, generic solutions for systematically developing adaptive loops were missing. Motivated by this fact, we have developed HAFLoop. HAFLoop is a generic and reusable solution that easies and fastens the design and implementation of adaptive AMs. Our solution enables AMs to support structural and parameter adaptation of its components as well as organize them in a variety of settings, from centralized to fully decentralized. The adaptation process of the feedback loops implementing SASs' AMs is driven by a set of runtime policies, which describe structural and behavioral aspects of the AM components.

HAFLoop has been evaluated in two application domains: SDVs and IoT networks. The evaluation results have demonstrated not only the feasibility of applying HAFLoop in modern, extremely demanding SASs, but also the benefits of adopting adaptive feedback

loops in these domains. In future work, we plan to apply HAFLoop in real systems and test different use cases. For instance, adapt the rest of the elements of the loop. For the SDV, we also plan to run experiments using different data mining algorithms, while for the IoT network we plan to investigate their incorporation into the loop. Finally, our work could be extended by running experiments with different values on the policy variables, e.g., using different values for the number of iterations to wait before triggering the need for adaptation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Thanks to CONACYT, Mexico, for the PhD scholarship granted to Edith Zavala. Also, thanks to the vehicle laboratory Revere for the technical support provided during HAFLoop evaluation. This work has been partially supported by the AstaZero, Sweden openresearch@astazero program (call 4 - 20180430); and the Spanish project GENESIS (TIN2016-79269-R).

Appendix

See Tables A.1–A.4.

References

[1] C. Krupitzer, F.M. Roth, S. Vansyckel, G. Schiele, C. Becker, A survey on engineering approaches for self-adaptive systems, Pervasive Mob. Comput. 17 (2015) 184–206, http://dx.doi.org/10.1016/j.pmcj.2014.09.009.

[2] B.H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H.M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H.A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, J. Whittle, Software engineering for self-adaptive systems: A research roadmap, Softw. Eng. Self-Adapt. Syst. (2009) 1–26, http://dx.doi.org/10.1007/978-3-642-02161-9_2.

[3] R. De Lemos, H. Giese, H.A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N.M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K.M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D.B. Smith, J.P. Sousa, L. Tahvildari, K. Wong, J. Wuttke, Software engineering for self-adaptive systems: A second research roadmap, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), in: LNCS, vol. 7475, 2013, pp. 1–32, http://dx.doi.org/10.1007/978-3-642-35813-5_1.

[4] D. Weyns, Software engineering of self-adaptive systems: An organised tour and future challenges, Handb. Softw. Eng. (2017) 1–41.

[5] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, K.M. Göschka, On patterns for decentralized control in self-adaptive systems, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), in: LNCS, vol. 7475, 2013, pp. 76–107, http://dx.doi.org/10.1007/978-3-642-35813-5_4.

[6] BMI, An architectural blueprint for autonomic computing, IBM White Pap. 36 (2006) 34, http://dx.doi.org/10.1021/am900608j.

[7] J.O. Kephart, D.M. Chess, The vision of autonomic computing, IEEE Comput. Soc. 36 (2003) 41–50, http://dx.doi.org/10.1109/MC.2003.1160055.

[8] R.J. Anthony, M. Pelc, W. Byrski, Context-aware reconfiguration of autonomic managers in real-time control applications, in: Proceeding 7th Int. Conf. Auton. Comput. - ICAC '10, ACM, New York, NY, USA, 2010, pp. 73–74, http://dx.doi.org/10.1145/1809049.1809061.

[9] M.U. Iftikhar, D. Weyns, Assuring system goals under uncertainty with active formal models of self-adaptation, in: Companion Proc. 36th Int. Conf. Softw. Eng. - ICSE Companion 2014, ACM, New York, NY, USA, 2014, pp. 604–605, http://dx.doi.org/10.1145/2591062.2591137.

[10] C. Krupitzer, J. Otto, F.M. Roth, A. Frommgen, C. Becker, Adding self-improvement to an autonomic traffic management system, in: Proc. - 2017 IEEE Int. Conf. Auton. Comput. ICAC 2017, IEEE, 2017, pp. 209–214, http://dx.doi.org/10.1109/ICAC.2017.16.

[11] G. Tamura, N.M. Villegas, H.A. Muller, L. Duchien, L. Seinturier, Improving context-awareness in self-adaptation using the DYNAMICO reference model, in: 2013 8th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst., IEEE, 2013, pp. 153–162, http://dx.doi.org/10.1109/SEAMS.2013.6595502.

[12] T. Zhao, W. Zhang, H. Zhao, Z. Jin, A reinforcement learning-based framework for the generation and evolution of adaptation rules, in: Proc. - 2017 IEEE Int. Conf. Auton. Comput. ICAC 2017, IEEE, 2017, pp. 103–112, http://dx.doi.org/10.1109/ICAC.2017.47.

[13] F.M. Roth, C. Krupitzer, C. Becker, Runtime evolution of the adaptation logic in self-adaptive systems, in: Proc. - IEEE Int. Conf. Auton. Comput. ICAC 2015, IEEE, 2015, pp. 141–142, http://dx.doi.org/10.1109/ICAC.2015.20.

[14] E. Zavala, X. Franch, J. Marco, Adaptive monitoring: A systematic mapping, Inf. Softw. Technol. 105 (2019) 161–189, http://dx.doi.org/10.1016/j.infsof.2018.08.013.

[15] C. Berger, From a competition for self-driving miniature cars to a standardized experimental platform: Concept, models, architecture, and evaluation, J. Softw. Eng. Robot. 5 (2014) 63–79.

[16] M.U. Iftikhar, G.S. Ramachandran, P. Bollansée, D. Weyns, D. Hughes, DeltaIoT: A self-adaptive internet of things exemplar, in: Proc. - 2017 IEEE/ACM 12th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst. SEAMS 2017, 2017, pp. 76–82, http://dx.doi.org/10.1109/SEAMS.2017.21.

[17] E. Zavala, X. Franch, J. Marco, A. Knauss, D. Damian, SACRE: Supporting contextual requirements' adaptation in modern self-adaptive systems in the presence of uncertainty at runtime, Expert Syst. Appl. 98 (2018) http://dx.doi.org/10.1016/j.eswa.2018.01.009.

[18] C. Krupitzer, F.M. Roth, S. Vansyckel, C. Becker, Towards reusability in autonomic computing, in: Proc. - IEEE Int. Conf. Auton. Comput. ICAC 2015, IEEE, 2015, pp. 115–120, http://dx.doi.org/10.1109/ICAC.2015.21.

[19] C. Krupitzer, S. Vansyckel, C. Becker, FESAS: Towards a framework for engineering self-adaptive systems, in: 2013 IEEE 7th Int. Conf. Self-Adaptive Self-Organizing Syst., IEEE, 2013, pp. 263–264, http://dx.doi.org/10.1109/SASO.2013.36.

[20] J. Kramer, J. Magee, Self-managed systems: an architectural challenge, in: Futur. Softw. Eng. (FOSE '07), IEEE, 2007, pp. 259–268, http://dx.doi.org/10.1109/FOSE.2007.19.

[21] M.U. Iftikhar, D. Weyns, ActivFORMS: Active formal models for self-adaptation, in: Proc. 9th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst., 2014, pp. 125–134, http://dx.doi.org/10.1145/2593929.2593944.

[22] D. Weyns, M.U. Iftikhar, ActivFORMS: A model-based approach to engineer self-adaptive systems, X, 2019, http://arxiv.org/abs/1908.11179.

[23] N. Medvidovic, D.S. Rosenblum, R.N. Taylor, A language and environment for architecture-based software development and evolution, in: Proc. 21st Int. Conf. Softw. Eng. - ICSE '99, 1999, pp. 44–53, http://dx.doi.org/10.1145/302405.302410.

[24] H. Tajalli, J. Garcia, G. Edwards, N. Medvidovic, PLASMA: A plan-based layered architecture for software model-driven adaptation, in: Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng. - ASE '10, ACM Press, New York, New York, USA, 2010, p. 467, http://dx.doi.org/10.1145/1858996.1859092.

[25] C. Dorn, S. Dustdar, Interaction-driven self-adaptation of service ensembles, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), in: LNCS, vol. 6051, 2010, pp. 393–408, http://dx.doi.org/10.1007/978-3-642-13094-6_31.

[26] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, S. Uchitel, MORPH: a reference architecture for configuration and behaviour self-adaptation, 2015, http://dx.doi.org/10.1145/2804337.2804339.

[27] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, S. Uchitel, An extended description of MORPH: A reference architecture for configuration and behaviour self-adaptation, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), in: LNCS, vol. 9640, 2017, pp. 377–408, http://dx.doi.org/10.1007/978-3-319-74183-3_13.

[28] I. Gerostathopoulos, T. Bures, P. Hnetynka, A. Hujecek, F. Plasil, D. Skoda, Strengthening adaptation in cyber-physical systems via meta-adaptation strategies, ACM Trans. Cyber-Phys. Syst. 1 (2017) 1–25, http://dx.doi.org/10.1145/2823345.

[29] I. Gerostathopoulos, D. Skoda, F. Plasil, T. Bures, A. Knauss, Tuning self-adaptation in cyber-physical systems through architectural homeostasis, J. Syst. Softw. 148 (2019) 37–55, http://dx.doi.org/10.1016/j.jss.2018.10.051.

[30] G. Perrouin, B. Morin, F. Chauvel, F. Fleurey, J. Klein, Y. Le Traon, O. Barais, J.M. Jezequel, Towards flexible evolution of dynamically adaptive systems, in: Proc. Int. Conf. Softw. Eng., 2012, pp. 1353–1356, http://dx.doi.org/10.1109/ICSE.2012.6227081.

[31] A. Knauss, D. Damian, X. Franch, A. Rook, H.A. Müller, A. Thomo, Acon: A learning-based approach to deal with uncertainty in contextual requirements at runtime, Inf. Softw. Technol. 70 (2016) 85–99, http://dx.doi.org/10.1016/j.infsof.2015.10.001.

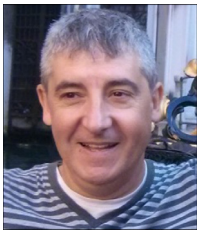
[32] E. Zavala, X. Franch, J. Marco, A. Knauss, D. Damian, SACRE: A tool for dealing with uncertainty in contextual requirements at runtime, in: 23rd IEEE Int. Requir. Eng. Conf, IEEE, 2015, pp. 278–279, http://dx.doi.org/10.1109/RE.2015.7320437.

[33] J. Branke, P. Goldate, H. Prothmann, Actuated traffic signal optimization using evolutionary algorithms, in: Proc. 6th Eur. Congr. Exhib. Intell. Transp. Syst. Serv. (ITS 2007), 2007, http://dx.doi.org/10.1179/1743284713Y.0000000425.

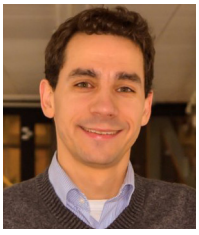
[34] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, H. Schmeck, Organic computing - Addressing complexity by controlled self-organization, in: Proc. - ISoLA 2006 2nd Int. Symp. Leveraging Appl. Form. Methods, Verif. Valid., 2007, pp. 185–191, http://dx.doi.org/10.1109/ISoLA.2006.19.
[35] H. Prothmann, F. Rochner, S. Tomforde, J. Branke, C. Müller-Schloer, H. Schmeck, Organic control of traffic lights, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), in: LNCS, vol. 5060, 2008, pp. 219–233, http://dx.doi.org/10.1007/978-3-540-69295-9_19.
[36] F. Rochner, H. Prothmann, J. Branke, C. Müller-Schloer, H. Schmeck, An organic architecture for traffic light controllers, Proc. Inform. 1 (2006) 120–127.
[37] S. Tomforde, H. Prothmann, F. Rochner, J. Branke, J. Hähner, C. Müller-Schloer, H. Schmeck, Decentralised progressive signal systems for organic traffic control, in: Proc. - 2nd IEEE Int. Conf. Self-Adaptive Self-Organizing Syst. SASO 2008, 2008, pp. 413–422, http://dx.doi.org/10.1109/SASO.2008.31.
[38] D. Sykes, W. Heaven, J. Magee, J. Kramer, From goals to components: a combined approach to self-management, in: Proc. 2008 Int. Work. Softw. Eng. Adapt. Self-Managing Syst. - SEAMS '08, ACM Press, New York, New York, USA, 2008, p. 1, http://dx.doi.org/10.1145/1370018.1370020.
[39] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, K. Inoue, Learning revised models for planning in adaptive systems, in: 2013 35th Int. Conf. Softw. Eng., IEEE, 2013, pp. 63–71, http://dx.doi.org/10.1109/ICSE.2013.6606552.
[40] D. Sykes, J. Magee, J. Kramer, FlashMob: Distributed adaptive self-assembly, in: Proceeding 6th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst. - SEAMS '11, 2011, p. 100, http://dx.doi.org/10.1145/1988008.1988023.
[41] I. Epifani, C. Ghezzi, R. Mirandola, G. Tamburrelli, Model evolution by run-time parameter adaptation, in: Proc. - Int. Conf. Softw. Eng., IEEE, 2009, pp. 111–121, http://dx.doi.org/10.1109/ICSE.2009.5070513.
[42] A. Elkhodary, N. Esfahani, S. Malek, FUSION: A framework for engineering self-tuning self-adaptive software systems, in: Proc. ACM SIGSOFT Symp. Found. Softw. Eng., 2010, pp. 7–16, http://dx.doi.org/10.1145/1882291.1882296.
[43] N. Esfahani, A. Elkhodary, S. Malek, A learning-based framework for engineering feature-oriented self-adaptive software systems, IEEE Trans. Softw. Eng. 39 (2013) 1467–1493, http://dx.doi.org/10.1109/TSE.2013.37.
[44] P. Jamshidi, A.M. Sharifloo, C. Pahl, A. Metzger, G. Estrada, Self-learning cloud controllers: Fuzzy Q-learning for knowledge evolution, in: Proc. - 2015 Int. Conf. Cloud Auton. Comput. ICCAC 2015, 2015, pp. 208–211, http://dx.doi.org/10.1109/ICCAC.2015.35.
[45] C. Quinton, R. Rabiser, M. Vierhauser, P. Grünbacher, L. Baresi, Evolution in dynamic software product lines: challenges and perspectives, in: Proc. 19th Int. Conf. Softw. Prod. Line - SPLC '15, 2015, pp. 126–130, http://dx.doi.org/10.1145/2791060.2791101.
[46] A.M. Sharifloo, A. Metzger, C. Quinton, L. Baresi, K. Pohl, Learning and evolution in dynamic software product lines, in: Proc. 11th Int. Work. Softw. Eng. Adapt. Self-Managing Syst. - SEAMS '16, ACM Press, New York, New York, USA, 2016, pp. 158–164, http://dx.doi.org/10.1145/2897053.2897058.
[47] L. Baresi, C. Quinton, Dynamically evolving the structural variability of dynamic software product lines, in: Proc. - 10th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst. SEAMS 2015, 2015, pp. 57–63, http://dx.doi.org/10.1109/SEAMS.2015.24.
[48] V. Klos, T. Gothel, S. Glesner, Adaptive knowledge bases in self-adaptive system design, in: Proc. - 41st Euromicro Conf. Softw. Eng. Adv. Appl. SEAA 2015, 2015, pp. 472–478, http://dx.doi.org/10.1109/SEAA.2015.48.
[49] A. Rodrigues, R.D. Caldas, G.N. Rodrigues, T. Vogel, P. Pelliccione, A learning approach to enhance assurances for real-time self-adaptive systems, 2018, pp. 206–216, http://dx.doi.org/10.1145/3194133.3194147.
[50] G.G. Pascual, M. Pinto, L. Fuentes, Self-adaptation of mobile systems driven by the common variability language, Future Gener. Comput. Syst. 47 (2015) 127–144, http://dx.doi.org/10.1016/j.future.2014.08.015.
[51] Z.A. Mann, A. Metzger, Auto-adjusting self-adaptive software systems, in: 2018 IEEE Int. Conf. Auton. Comput., 2018, pp. 181–186, http://dx.doi.org/10.1109/ICAC.2018.00030.
[52] R.J. Anthony, Policy-centric integration and dynamic composition of autonomic computing techniques, in: ICAC, 2007, http://dx.doi.org/10.1109/ICAC.2007.32.
[53] R.J. Anthony, A versatile policy toolkit supporting run-time policy reconfiguration, Cluster Comput. 11 (2008) 287–298, http://dx.doi.org/10.1007/s10586-008-0058-7.
[54] R. Anthony, D. Chen, M. Törngren, D. Scholle, M. Sanfridson, A. Rettberg, T. Naseer, M. Persson, L. Feng, Autonomic middleware for automotive embedded systems, in: Auton. Commun., 2009, pp. 169–210, http://dx.doi.org/10.1007/978-0-387-09753-4_7.
[55] N.M. Villegas, G. Tamura, H.A. Müller, L. Duchien, R. Casallas, DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), in: LNCS, vol. 7475, 2013, pp. 265–293, http://dx.doi.org/10.1007/978-3-642-35813-5_11.
[56] E. Lee, Y.-G.G. Kim, Y.-D.D. Seo, K. Seol, D.-K.K. Baik, RINGA: Design and verification of finite state machine for self-adaptive software at runtime, Inf. Softw. Technol. 93 (2018) 200–222, http://dx.doi.org/10.1016/j.infsof.2017.09.008.
[57] H. Nakagawa, A. Ohsuga, S. Honiden, Towards dynamic evolution of self-adaptive systems based on dynamic updating of control loops, in: 2012 IEEE Sixth Int. Conf. Self-Adaptive Self-Organizing Syst., 2012, pp. 59–68, http://dx.doi.org/10.1109/SASO.2012.17.
[58] L. Baresi, L. Pasquale, An eclipse plug-in to model system requirements and adaptation capabilities, in: Proc. 6th IT-Eclipse Work, 2011.
[59] L. Baresi, L. Pasquale, P. Spoletini, Fuzzy goals for requirements-driven adaptation, in: Proc. 2010 18th IEEE Int. Requir. Eng. Conf., 2010, pp. 125–134, http://dx.doi.org/10.1109/RE.2010.25.
[60] L. Pasquale, L. Baresi, B. Nuseibeh, Towards adaptive systems through requirements@runtime?, in: CEUR Workshop Proc., 2011, pp. 13–24.
[61] N. Ali, C. Solis, Self-adaptation to mobile resources in service oriented architecture, in: Proc. - 2015 IEEE 3rd Int. Conf. Mob. Serv. MS 2015, 2015, pp. 407–414, http://dx.doi.org/10.1109/MobServ.2015.62.
[62] C. Krupitzer, F.M. Roth, S. Vansyckel, G. Schiele, C. Becker, A survey on engineering approaches for self-adaptive systems, Pervasive Mob. Comput. 17 (2015) 184–206, http://dx.doi.org/10.1016/j.pmcj.2014.09.009.
[63] V. Chang, M. Abdel-Basset, M. Ramachandran, Towards a reuse strategic decision pattern framework – from theories to practices, Inf. Syst. Front. 21 (2019) 27–44, http://dx.doi.org/10.1007/s10796-018-9853-8.
[64] P. Fremantle, B. Aziz, Deriving event data sharing in IoT systems using formal modelling and analysis, Internet Things 8 (2019) 100092, http://dx.doi.org/10.1016/j.iot.2019.100092.
[65] F. Al-Turjman, M. Abujubbeh, IoT-enabled smart grid via SM: An overview, Future Gener. Comput. Syst. 96 (2019) 579–590, http://dx.doi.org/10.1016/j.future.2019.02.012.
[66] R. Han, G. Shapiro, V. Gramoli, X. Xu, On the performance of distributed ledgers for internet of things, Internet Things (2019) 100087, http://dx.doi.org/10.1016/j.iot.2019.100087.
[67] F. Al-Turjman, A. Malekloo, Smart parking in IoT-enabled cities: A survey, Sustain. Cities Soc. 49 (2019) http://dx.doi.org/10.1016/j.scs.2019.101608.
[68] A. Rook, A. Knauss, D. Damian, A. Thomo, A case study of applying data mining to sensor data for contextual requirements analysis, in: 2014 IEEE 1st Int. Work. Artif. Intell. Requir. Eng., 2014, pp. 43–50, http://dx.doi.org/10.1109/AIRE.2014.6894855.
[69] A. Rook, On the Feasibility of Integrating Data Mining Algorithms into Self Adaptive Systems for Context Awareness and Requirements Evolution (Master thesis), University of Victoria, 2014.
[70] M.L. Berenson, D.M. Levine, Basic Bussiness Statistics: Concepts and Applications, sixth ed., Prentice-Hall International, Inc., 1996.

Edith Zavala received her Ph.D. degree in Computing from Universitat Politècnica de Catalunya (UPC-BarcelonaTech), Spain, in 2019. She is member of the Software and Service Engineering research group (GESSI) of the UPC-BarcelonaTech since 2015. Currently, she is working in the fields of Self-adaptive systems and Adaptive monitoring. She is especially focused on the research of engineering solutions for intelligent systems and the integration of Machine learning techniques in such type of systems. Her research interests are in software architecture, distributed systems, runtime adaptive applications and the engineering of Artificial Intelligence for such complex systems.

Xavier Franch received his Ph.D. degree in Informatics from Universitat Politècnica de Catalunya (UPC-BarcelonaTech), Spain, in 1996. Currently, he is a professor in Software Engineering at the UPC-BarcelonaTech. His research interest embraces many fields in software engineering, including requirements engineering, empirical software engineering, open source software, and agile software development. Prof. Franch is a member of the IST, REJ, IJCIS, and Computing editorial boards, Journal First chair of JSS, and Deputy Editor of IET Software. He served as PC chair at RE'16, ICSOC'14, CAiSE'12, and REFSQ'11, among others, and as General Chair for RE'08 and PROFES'19. More information at https://www.essi.upc.edu/~franch.



Jordi Marco received the M.Sc. and Ph.D. degrees in computing from Universitat Politècnica de Catalunya (UPC-BarcelonaTech), Spain, in 2005. Currently, he is an Associate Professor in Computer Science at the UPC-BarcelonaTech. His research interests include service-oriented computing, quality of service, conceptual modeling, container libraries, and computer graphics. Dr. Marco has been PC member on several international conferences like ATSE, QASBA, RCIS, and BIGDSE. He also reviewed papers for journals including ESWA and IST. More information at https://www.cs.upc.edu/~jmarco.



Christian Berger is Associate Professor and Docent for Software Engineering at the Department of Computer Science and Engineering at University of Gothenburg. His research focuses on systematically architecting complex software and systems embracing continuous integration, continuous deployment, and continuous experimentation for a growingly automated and digitalized society. He is an expert for self-driving vehicles with over a decade of experience. Currently, he jointly leads the design and development of the open source platform Open Driverless Vehicle (OpenDLV), software that powers research projects at Chalmers vehicle laboratory Revere.