



HAL
open science

IAS: an IoT Architectural Self-adaptation Framework

Mahyar T Moghaddam, Eric Rutten, Philippe Lalanda, Guillaume Giraud

► **To cite this version:**

Mahyar T Moghaddam, Eric Rutten, Philippe Lalanda, Guillaume Giraud. IAS: an IoT Architectural Self-adaptation Framework. ECSA 2020 - 14th European Conference on Software Architecture, Sep 2020, L'Aquila, Italy. pp.1-16. hal-02900674

HAL Id: hal-02900674

<https://inria.hal.science/hal-02900674>

Submitted on 16 Jul 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IAS: an IoT Architectural Self-adaptation Framework

Mahyar T. Moghaddam¹, Eric Rutten¹, Philippe Lalanda², and Guillaume Giraud³

¹ Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble France

² Univ. Grenoble Alpes, LIG, F-38058 Grenoble France

³ RTE, F-92073 Paris France

{mahyar.tourchi-moghaddam, eric.rutten}@inria.fr

philippe.lalanda@imag.fr

guillaume-np.giraud@rte-france.com

Abstract. This paper develops a generic approach to model control loops and their interaction within the Internet of Things (IoT) environments. We take advantage of *MAPE-K* loops to enable architectural self-adaptation. The system’s architectural setting is aligned with the adaptation goals and the components run-time situation and constraints. We introduce an integrated framework for IoT Architectural Self-adaptation (*IAS*) where *functional* control elements are in charge of environmental adaptation and *autonomic* control elements handle the functional system’s architectural adaptation. A *Queuing Networks (QN)* approach was used for modeling the *IAS*. The *IAS-QN* can model control levels and their interaction to perform both architectural and environmental adaptations. The *IAS-QN* was modeled on a smart grid system for the Melle-Longchamp area (France). Our architectural adaptation approach successfully set the propositions to enhance the performance of the electricity transmission system. This industrial use-case is a part of *CPS4EU* European industrial innovation project.¹

Keywords: IoT · Software architecture · Self-adaptation · Autonomic control · Functional control · Performance · Queuing networks.

1 Introduction

Internet of Things (*IoT*) systems are composed of distributed smart elements that are pervasively installed to affect the environment. Like most software systems, *IoT* is exposed to changes that occur in both their state and their surrounding environment. The changes cause uncertainties during system operation. Control loops are introduced to facilitate self-adaptation to handle changes and uncertainties. *IoT* sensors supply raw data (*M*) to central or distributed computational components to be refined and analyzed (*A*) towards further actuation planning (*P*) and execution (*E*). This process within comprehensive knowledge (*K*) forms the *MAPE-K* control loop. Control loops can be designed and developed in many different ways. Architecture-based adaptation is an example that focuses on the role of architectures in engineering self-adaptive systems. Typically, modeling architectural self-adaptation imposes separating the concerns about system functionality from adaptation [1].

¹ CPS4EU is a three years project funded by the H2020-ECSEL-2018-IA. The project develops four vital IoT technologies, namely computing, connectivity, sensing, and cooperative systems. It incorporates those IoT technologies through pre-integrated architectures and design tools. It instantiates the architectures in dedicated use-cases from a strategic application viewpoint for automotive, smart grid, and industrial automation. <https://cps4eu.eu>

In contrast to most of the architecture-based adaptation models, we propose an approach that considers the adaptation internal to the system functionality. More specifically, we regard functional control elements (*FCE*) in charge of managing the system functionality and autonomic control elements (*ACE*) responsible for monitoring the functional system’s situation and handling the architectural composition. In our **IoT Architectural Self-adaptation (IAS)** framework, we are concerned with the interaction among various levels of control loops that are driven by the system adaptation goals. Our focus is on reasoning and modeling various *IoT* architectural patterns and their run-time architectural transitions managed by the autonomic control logic. The *IAS* conceptual framework, while inspired by the *IEEE/ISO/IEC 42010* architecture description standard [2], comprises both functional and autonomic control elements as well as their interaction mechanisms.

We define the *IAS* conceptual framework, and we model it on a real smart grid application: the Melle-Longchamp area (France). Since the area expands renewable energy generation using several wind-farms as sources of energy, the voltage and current of the system sometimes become hard to forecast. Therefore, to avoid the risk of overloading the lines and creating danger for people’s safety, the peak current has to be managed. Instead of developing new installations, the French Transmission System Operator policy is to investigate new exploitation methods of the existing electrical installations and favor their optimal operation. Wind-farm generation can be limited by opening their feeder’s circuit breaker, or more efficiently, by modulating their generation. Additional means can also be used, such as batteries, power electronics, and *IoT*. The heterogeneity and variation of sensors, actuators, and processing elements of power systems increase the concerns on reliability and performance. In our use-case, while the circuit breakers are the safe and quick solution to avoid overloading of lines, their usage should be minimized to prevent imposing indirect costs. Modulating wind-farms’ generation is a solution exposed to a high actuation time, and batteries can store electricity for a few seconds. Thus, the system needs to make quick decisions on its own composition to keep the performance within an adequate threshold.

Putting the self-adaptation control at the center of the software process, we started by analyzing the problem and selecting the data to see what factors affect the system response time. Then we upgraded the software architecture from local centralized to hierarchical, which enables all types of architectural transition. We further modeled the *IAS* approach by queuing networks (QNs) that facilitate designing the various levels of control for performance evaluation.

The paper makes the following contributions: *i*) presenting an IoT architectural self-adaptation framework that focuses on functional and autonomic control components and their run-time interaction; *ii*) modeling the proposed framework with queuing networks to estimate the performance of IoT systems and to support architectural decisions and transitions; *iii*) applying our framework to a smart grid system by analyzing its various components and their run-time behavior, for establishing performant operations.

The paper is organized as follows. Relevant literature is discussed in Section 2. The *IAS* framework is thoroughly explained in Section 3. The approach is applied to a real case in Section 4, and conclusions are finally drawn in Section 5.

2 Related Work

In software engineering, works on self-adaptation typically focus on functional control elements that interact with the environment to provide a service. Here we find works on using feedback control loops (such as *MAPE-K*) and their interaction that can be presented as patterns [3], in which the functions from multiple loops are coordinated in different ways. Such interactive coordination

mechanisms are indeed crucial to model ever-growing distributed systems. Each interaction pattern can satisfy several non-functional requirements while guaranteeing the functionality of the system [4]. To quote an example, *QoS*MOS [5] is an adaptive service-based platform that enables dynamic adaptation to run-time changes to achieve some quality of service (*QoS*) requirements. Some studies [6] take advantage of layered queuing networks (*LQNs*) while considering run-time *QoS* to automatically generate adaptation policies. Each element of *MAPE-K* loop should dynamically react [7] to changes that occur in system's goals and requirements. Current research on goal modeling takes into account uncertainty [8], but the goals' dynamic transition [1] and multiple dynamic goals' satisfaction [9] has not received much attention. We believe that the self-adaptive software systems' goals are highly influenced by the limitations and constraints imposed by the non-controllable environment. Various modes of functional requirement satisfaction should be engineered to enable the system to pick, synthesize, and verify those modes dynamically.

Such a challenge is even bolder in IoT systems, which comprise heterogeneous devices that dynamically interact through the internet. The problem can be tackled by designing self-managing devices that can adapt their state to changes in the system context and environment [10] [11]. However, realizing the IoT devices is challenging because of inherent uncertainties in their operation contexts, such as interferences and dynamic traffic in the network [12]. Often these uncertainties are difficult to predict by architects at development time and often lead to indecisiveness.

Several studies propose the use of software architectures to address self-adaptation [13] [14]. An architecture model provides a global view of the system and its properties and behavior [15]. While architectures can give a global idea of the system, the heterogeneity of software systems makes it challenging to design a set of self-adaptation architectural patterns. Some studies argue that architectural adaptation includes an architectural model of the controllable software components that allows the feedback loop to reason about various system configurations and adapt it based on goals [16]. However, considering the feedback loop running on *FCE* as an external mechanism to the system minimizes the dynamicity of the self-adaptive system. We believe that the functional control mechanism should be monitored and adapted by autonomic control components [17] [18], which gets input from both dynamic goals and real-time state of the system.

3 IAS Framework

This section introduces the conceptual foundations of *IAS*, comprising a metamodel that focuses on the *FCE* and *ACE* interaction. The framework is inspired by the *IEEE/ISO/IEC 42010* standard [2], but focuses on architecture self-adaptation rather than architecture description. The metamodel (Figure 1) depicts vital concepts of systems and the control mechanism as a process to be considered in the software design and adaptation process. The metamodel is divided into two parts: the right side depicts *functional control* component and its inputs and dependencies, and the left side deals with *autonomic control* component and its correlation with other elements of the system. Software system **stakeholders** comprise users, developers/clients/managers, and citizens/occupants. Stakeholders have concerns regarding the system-of-interest [2]. As the focus of this paper, developers and managers are concerned with the **architecture variant**, including the life cycle from system needs and requirements, design choices and implementation, and operating considerations.

IEEE/ISO/IEC 42010 standard [2] specifies that the system goals and concerns are traditionally formed of functional and non-functional requirements, design constraints, assumptions, dependencies, and architecture decisions. A system contains both **functional** and **adaptation goals** that are set by stakeholders. Functional goals specify the system's functionality under various environmental

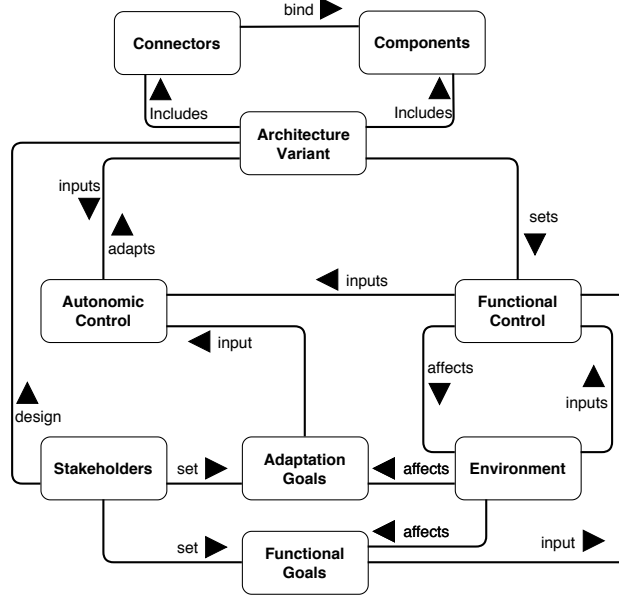


Fig. 1. Conceptual model of IoT Architectural Self-adaptation (IAS).

constraints, and adaptation goals mostly concern the quality of the system. In the *IAS* approach, we argue that self-adaptation is a goal-directed process and its goals should be captured. As shown in Figure 1, the goals are generally affected by the **environment**. In other words, the environment context might enforce prioritizing a set of goals or ignoring another set of goals. For instance, if the goal of a self-adaptive smart grid system is performance improvement, a disaster may prioritize taking adequate measures to prevent the emergency by, e.g., activating circuit breakers.

Thus, a system is situated in the environment. The environment is the real world, by which the software system interacts. The environment might include both physical and virtual elements [1], that the system does not directly control their functionality. The system interacts with the environment and is influenced by it. A system can also interact with other systems in the environment.

The environment can be sensed and affected through *sensors* and *actuators*, respectively, which locate within *IoT elements* subsystem and perform the functionality of the IoT system. As shown in Figure 2/right, the *sense* elements frequently retrieve raw data [19] to input the control components, and *actuate* elements receive periodic commands to affect environment. The mentioned data transmission is continuous since the environment is not under full control of the software system, and the dynamics of the environment should be tackled.

The **functional control** comprises the adaptation logic that allows the system to perform the intended adaptation within the environment. The *FCE* has a *MAPE-K* (*Monitor, Analysis, Plan, Execute and comprehensive knowledge*) approach behind [17] [19] [14]. The *Monitor* element aggregates and refines the data to be analyzed and updates the *knowledge* base of the control component. The *Analyze* element interprets the monitored data based on the *functional goals*. The *Plan* element builds actuation strategies, and the *Execute* element processes the actuation strategies and prepares the type of message to be set to each set of actuators.

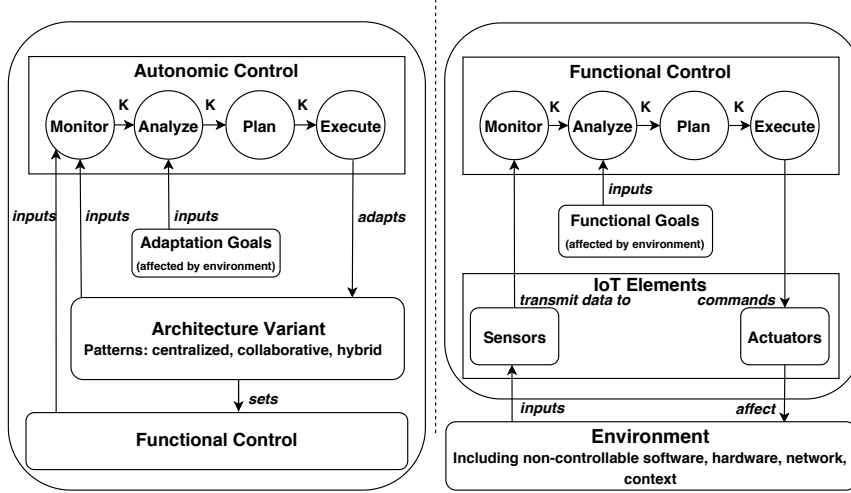


Fig. 2. IAS autonomous control (right) and functional control (left) mechanisms.

The left side of the metamodel (Figure 1) shows the **autonomic control** that is more extensively described within Figure 2/left. The autonomic control supports a continuous self-adaptation process [17]. It enables the system to monitor itself continuously and perform necessary adaptation to achieve the adaptation goals. The *ACE* takes advantage of the *MAPE-K* concept as well. It *monitors* the system's situation (including *functional control*) and assesses both the system functionality and quality to update the *knowledge* base. The *ACE* further *analyzes* the data and compares it with real-time adaptation goals. Afterward, an adequate strategy will be *planned* to be *executed* by architecture variant adaptation. For instance, suppose that the adaptation goal is to keep the performance in a proper threshold, and the high CPU time on a local controller is preventing such a purpose. In this situation, the autonomic control component adapts the architecture based on a specific strategy, e.g., switching from local to the remote control.

Architecture variant determines variations in both software and hardware architectures [20]. The hardware architecture includes *IoT* hardware elements, i.e., sensors, network facilities, controllers, and actuators. The software architecture that is run on hardware elements includes a set of **components** that are bounded by **connectors** based on specific rules and constraints. These architectures are designed by stakeholders and self-adapted by *ACE* during system execution [21].

It is worth mentioning that, from a software architecture point of view, *FCE* and *ACE* and *architecture variant* are all part of the architecture. Architecture variant determines multiple functional deployment types, which appear as architectural patterns shown in Figure 3. The patterns are composed of *IoT elements* layer and one or several *functional control* layers. The functional control can perform locally and/or centrally and remotely. Here is the point in which a centralized cloud and distributed edge and fog can form the *hierarchical* pattern. Thus, the patterns [22] characterize *IoT* systems based on their levels of *distribution* and *collaboration* [22] [20]. Distribution specifies whether data analysis software ought to be deployed on a single node (*centralized*) or on several nodes (*distributed* and *hierarchical*) that are dispersed across the *IoT* system. The collaboration deals with interaction among functional control components to satisfy the goals, requirements, and

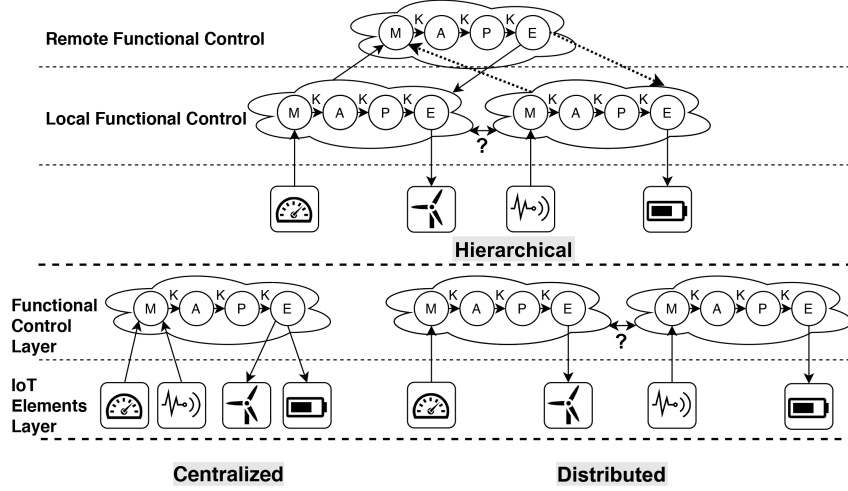


Fig. 3. *IoT* architectural patterns based on functional control components composition. The *centralized* pattern comprises processing on a *central local* or *remote* controller. The *distributed* pattern includes the processing on *independent* or *collaborative* controllers. The *Hierarchical* pattern contains *independent* or *hybrid* (i.e., with distributed collaborative) controllers.

strategies. This collaboration may appear as a level of information sharing, coordinated analysis or planning, or synchronized execution [14].

The *IAS-based* architectures contain the mechanisms to determine the required architectural adaption, based on intended *QoS* satisfaction level. Our conceptual framework does not rely on any specific tool; thus, practical modeling solutions can be mapped within it. The following section describes the steps taken to map a smart grid system within *IAS*, to improve its performance indices.

4 Application

We model our *IAS* framework on the performance improvement for *RTE² Company's* transmission network, located in the Melle-Longchamp area (France). Figure 4 shows the smart grid network that includes 35 substations connected by 30 lines. The grid has some constraints regarding current and voltage. In addition to the power flowing through the network, it contains wind-farms with a total peak production capacity of 700 MW. Melle-Longchamp area's control network is being upgraded from a traditional centralized control to an *IoT* distributed control system to enhance the performance of the software system. The system follows the usual sense - compute - actuate structure from *IoT* systems. We applied the *IAS* approach to analyze the system and its objectives, and to design an architectural self-adaptation mechanism that keeps the performance within a desirable threshold. It is worth mentioning that *IAS* and its associated generalized queuing networks models (*IAS-QN*) can be re-used for functionality and quality analysis of all *IoT* systems.

² Electricity Transmission Network, usually known as RTE, is the electricity transmission system operator of France.

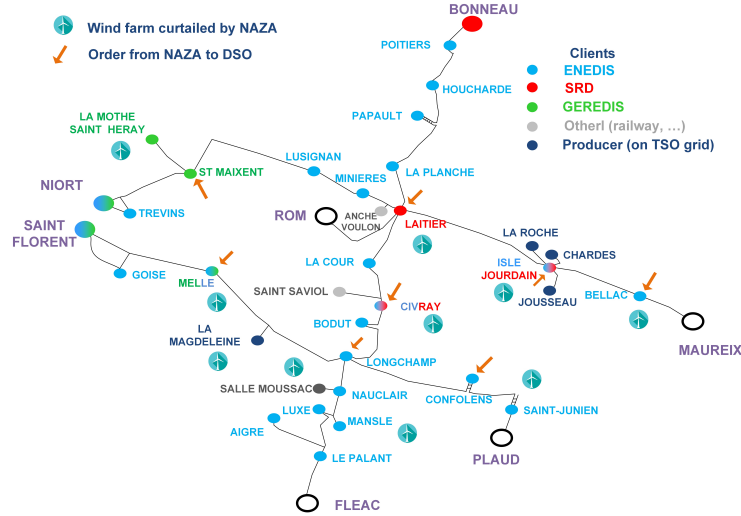


Fig. 4. Smart grid network for Melle-Longchamp. It includes 35 substations connected with 30 lines.

4.1 Problem and Goals Analysis

Renewable energy systems that convert wind and sun’s rays into electricity are growing as the primary source of energy. Such renewable generation is mainly connected to the distribution grid but has an impact on the transmission grid as well. In the example presented in Figure 5/lower, a high percentage of the required electricity to distribute is being supplied from RTE substation, e.g. B and a small percentage from distribution substation D . If a strong wind blows and the generation in D becomes excessive, an overload will occur on the transmission line between B and A . To deal with this problem, the functional controller can activate different levers: *i*) the battery in E can be charged, *ii*) the production in D can be limited, *iii*) the circuit breaker on B can be activated (less desirable option). Practically, a combination of the actions mentioned above is required. Dealing with transmission overload risk necessitates considering some information from sensors such as values of currents and voltages on every line, state of the network circuit breakers, state of battery’s charge, and also a set of parameters such as time to limit production of the wind farms, current overload thresholds on every line and eventually generator merit order.

Having such sensory input, FCE must ensure the safe operation of the network by sending: *i*) topological orders to the network circuit breakers, *ii*) modulation orders to the generators, *iii*) set-point orders to the storage batteries. The adaptive management of such smart transmission systems is exposed to performance issues since: *i*) some types of sensors and actuators need a significant service time, *ii*) enhanced forecasting algorithms for generation require a notable computation time, *iii*) network transmission and propagation delays sometimes become long, and *iv*) the collaboration pattern among local and remote control resources (with various processing power) is not always efficiently designed.

The typical application needs the delay between data acquisition and actuation to be less than *five seconds*, but shorter operation times seek. Within the next subsections, we design the RTE’s IAS-based system that enables the smart grid to tackle both functional and performance problems.

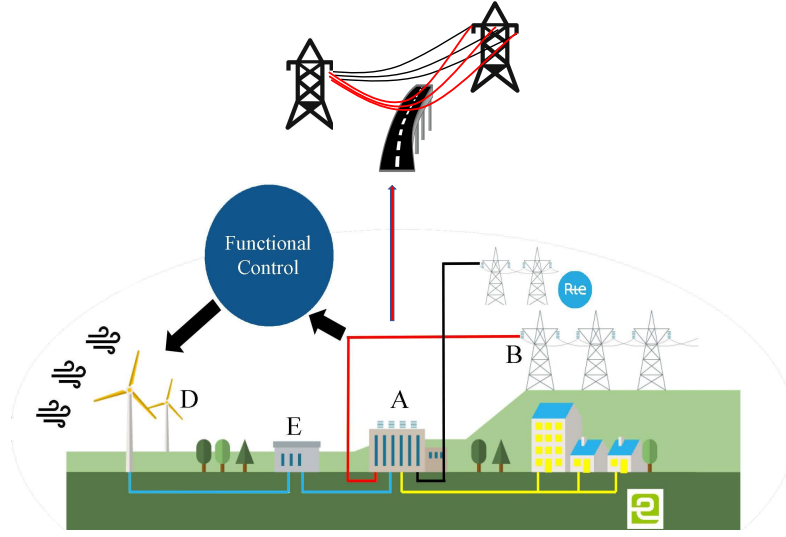


Fig. 5. The smart grid problem specification. The overloading of the lines because of e.g., a strong wind can create danger for people’s safety.

4.2 Architecture

Figure 6 shows the architecture we designed for the Melle-Longchamp case by taking advantage of the *New Automated Adaptive Zone (NAZA)* platform [23]. The architecture follows a hierarchical pattern with distributed collaborative controllers (see Figure 3) that can turn into centralized or distributed patterns if needed. As shown at the bottom of the figure, each of the 35 substations acquires data from two types of sensors: *i*) current and voltage transducers, and *ii*) position relays. This function can include aggregation or basic combination of acquired data (e.g., turning high-frequency sample values into root mean square values). Data from the sensors is sent to the control level, eventually after filtering. Each substation has a gateway that is in collaboration with other areas’ gateways. Gateways are servers *Advantech ECU-4787* or *MOXA 681-C*. Current and voltage measurements (in protocol *IEC61850*) are sent every second to the local gateway. The position of circuit breaker (in protocol *IEC60780-5-104*) is sent to the local gateway on every event.

Each gateway can act as the central controller of the whole network, with limited CPU capacity that is five times less than the central remote controller. The gateway on *substation 1* acts as the *autonomic control element (ACE)* that plans the combination of *functional control elements (FCE)* in use. The *ACE* can be moved to any other substation’s gateway or the cloud. The *ACE* implements the control logic given the states, conditions, and behaviors of the functional controllers.

As shown in the middle of Figure 6, the main functional element retrieves and stores data from the gateways, performs the computation, and sends orders to the actuators. *NAZA* platform principally relies on *RESTful API* to communicate with the transducers, relays, and actuators. The collected data is stored in a *MySQL DBMS*. The DBMS can also provide the solver inner-component with summary and real-time statistics. Besides, the system associated with the simulator service allows back-office to monitor the system state.

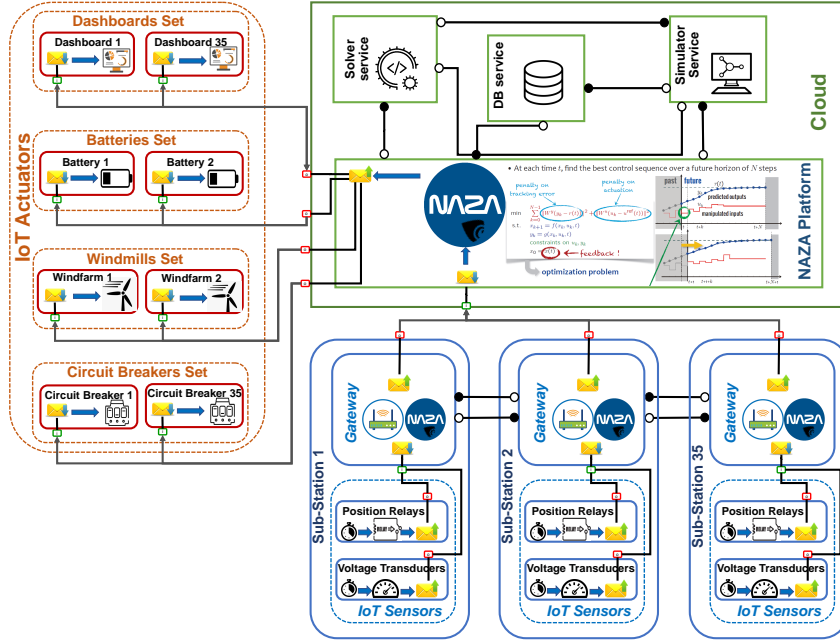


Fig. 6. The proposed hierarchical architecture for RTE. The architecture includes sense, process and actuating layers. *NAZA* platform can be run on gateways, local controller, and cloud.

The solver implements a Model Predictive Control (*MPC*) model to optimize a cost function to use levers such as batteries set-points and generation limit values. It gets real-time data from the gateways and calculates the values for actuators every 5 seconds. In some cases that the algorithm finds no solution or computation takes too long; simple flow charts enforce safety rules such as curtailing all necessary generations. *NAZA* platform can be run on RTE substation gateways, local controller, and cloud. Cloud has an unlimited processing power but causes 2 or 3 times more network delay than local servers. The left side of Figure 6 shows the actuators. Circuit breakers that are the fastest mode to stop the current in a line are located in every substation. The batteries can store the electricity for dozens of seconds to give some time to wind-farms to shut down. The dashboards show controllers' state, the values measured by sensors, and the set-points or limits sent to batteries and generators.

Our main architectural challenges are related to the the combination and location of the computation components, i.e., gateways, RTE controllers, and cloud. The challenge mainly arises when the intended *QoS* (here performance) is not satisfied, and a run-time architectural pattern switch is required. Such run-time dynamic adaptation and reconfiguration is set by *ACE*. The architectural patterns and their adaptation can be modeled by the Queuing Networks (*QNs*) concept. In the following section, we introduce a *QNs* modeling approach that can facilitate dealing with computation components' combination and location issues.

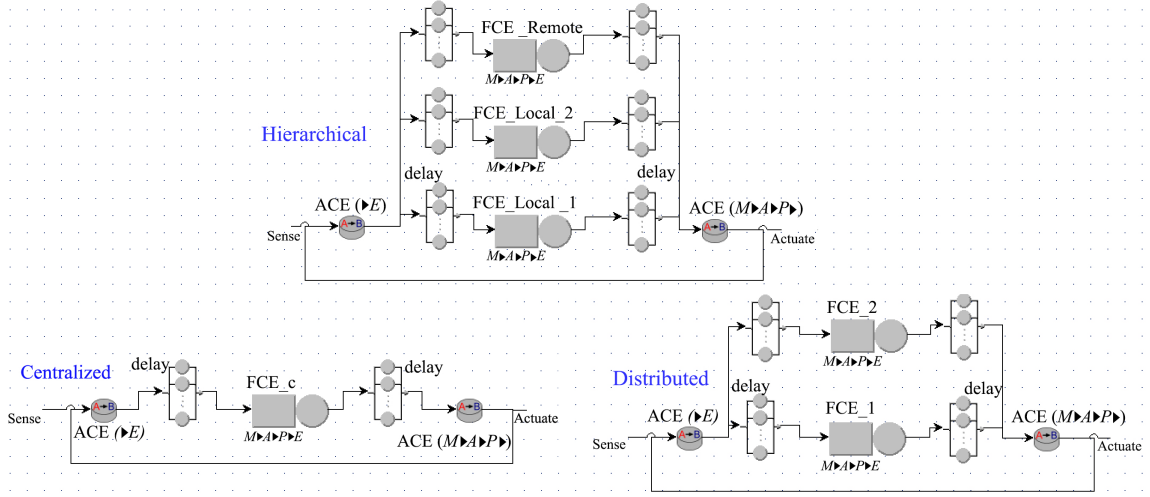


Fig. 7. IAS-QN patterns based on Figure 3. These generalized patterns can be re-used for functionality and quality analysis of all *IoT* systems.

4.3 Modeling IAS using QNs

In this section, we model the *IAS* within Queuing Networks (*QNs*) to introduce a generalized set of *IAS-QN* models. Our approach provides a pattern-based performance modeling of the entire self-adaptive system. The patterns can be re-used to model various self-adaptive *IoT* systems. In *IAS-QN*, the architectural components are represented by *QN* stations, and various sensing, computing, and actuating activities are represented by job classes that flow through the *QN*. In our *MAPEK-based* approach, the activities are performed both within and between components.

Figure 7 shows the *IAS-QN* patterns corresponding to the *IoT* architectural patterns shown in Figure 3. Data coming from *sense* elements feed the *controllers* to plan for specific *actuation*. The computation on sensed data is performed by the functional control elements (shown as *FCE*), while the composition of *FCE* is set by the autonomic control elements (shown as *ACE*). The *ACE* adopts a *MAPE-K* loop to assess the conformity between the *FCE* situation and the goals. Based on the locality of *ACE* and *FCE*, the communication between them suffers from some *delay*. *Centralized* pattern benefits from only one *FCE* so that the architectural adaptation can only take place on other elements, i.e., sense, actuate, and network. A centralized pattern can be associated with using a central server or cloud as the *FCE*.

Distributed pattern benefits from a minimum of two *FCE* that might share some information. Here the *ACE* can also enforce adaptation on *FCE* level by heading data toward a controller that enhances system quality. A distributed pattern is generally associated with the local processing and storage, which take place in *fog* nodes. Fog brings a degree of cloud functionality to the network edge. The computation capacity of fog is lower than the cloud, but it reduces a significant point of failure by shifting towards more than one computational component. However, fog only performs locally so that it does not have global coverage over large *IoT* systems. To tackle the mentioned shortcomings, *hierarchical* pattern that contains the advantages of both centralized and distributed

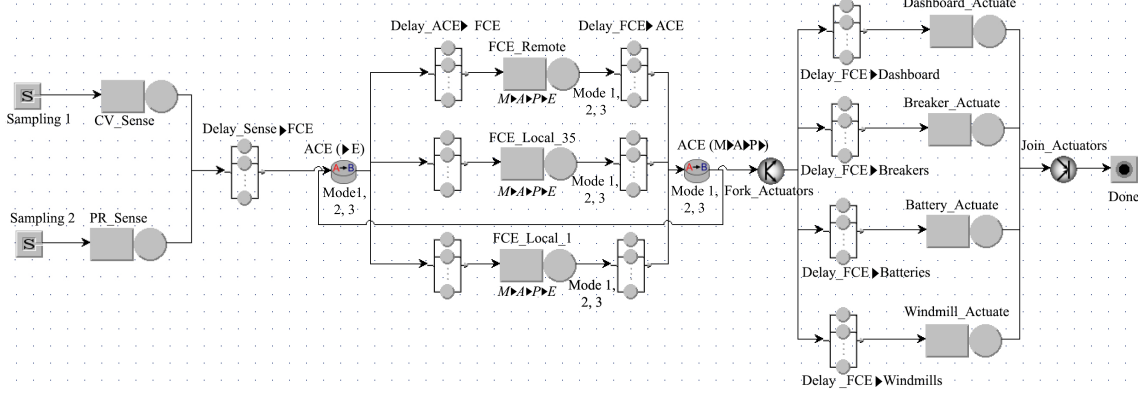


Fig. 8. *IAS-QN* for the Melle-Longchamp application. It conforms to the architecture shown in Figure 6. The case study consists of 35 local *FCE*, but 2 of them are shown here.

patterns is designed. In this pattern, the *ACE* can execute dynamic architectural adaptation by using local or remote functional controllers in a centralized or distributed way.

4.4 Modeling *IAS-QN* for the Melle-Longchamp application

Figure 8 depicts the *IAS-QN* designed for the Melle-Longchamp smart grid system. The case study consists of 35 local *FCE*, but because of the space limit, the figure shows 2 of them only. The proposed *IAS-QN* conforms the *hierarchical* architecture represented in Figure 6. However, it can dynamically switch to other patterns shown in Figure 7. The dynamic control flows through the *IAS-QN* components are specified as follows:

1. Environmental data are sampled by *sampling* nodes in charge of specifying the sampling rate. Our system has two types of sensors with different sampling rates; thus, two sampling nodes are required. The effect of the system on the environment is shown by *done* node. The mentioned nodes are located on the environment side (shown in Figure 2).
2. The *sense* nodes represent various types or clusters of sensors, which take as input specific kinds of sampled data. In our smart grid system, the transducers sample voltage and current every second, while the relays sample the circuit breakers position in an event-based manner. The sensory data is forwarded through a network to the controller(s). The network is exposed to both transmission delay (*td*) and propagation delay (*pd*).
3. The *autonomic control element (ACE)* is represented by *class-switches* nodes ($A \rightarrow B$) which use the *MAPE-K* logic. As we pointed out previously, the *ACE* class-switches, monitor the situation of the *FCE*, analyze its situation based on system goals, and plan for architectural adaptation execution in line with system goals (shown as $ACE: M \rightarrow A \rightarrow P$). The adaptation specifies where the sensory data should be routed for specific processing. When the feedback from the *FCE* goes back to the *ACE*, the composition of controllers to which the next job should be sent is decided (shown as $ACE: \rightarrow E$). Such decisions can be based on various control selection strategies:

- *Probabilities*: the destination controller is chosen according to predefined probabilities that, in general, are different for each controller.
- *Random*: the destination controller is chosen randomly; i.e., each controller has the same probability of selecting.
- *Round-Robin*: controllers are cyclically and circularly chosen as the destination controller.
- *Join the Shortest Queue*: the task is routed to the controller with the minimum number of tasks. The controller may be able to process the tasks immediately or with a queuing delay.
- *Shortest Response Time*: the task is instantly routed to the controller, which implies the minimum response time for the corresponding task type.
- *Least Utilization*: the controller with the smallest instant utilization is chosen as the destination controller.
- *Fastest Service*: the task is routed to the controller with the minimum service time for the corresponding task type.

Due to both the *RTE* preferences and the smart network configurations, we set the *ACE* logic based on *Probabilities* and keep the other strategies for future work. The following subsection clarifies the use of such probabilities.

4. The *Functional Control elements (FCE)* adopt the *MAPE-K* loop to achieve the system functional goals. In our smart grid system, the aggregated sensory data might be processed in distributed collaborative gateways and/or local or remote central controllers. The transition from local to central *FCE* (or vice versa) is a mode-switch dependent on the autonomic control, and the adoption of a more complex algorithmic model is a mode switch dependent on the functional control. These two levels of interactive adaptation drive the functionality of *IAS*-based systems. Mode transition in *IAS-QN* aims at adapting the control mechanisms and deployment of their execution by adequate actuation. In this example of application, a powerful *RTE* processor is used in the centralized pattern, while in the distributed pattern, the gateways with lower processing power collaborate to manage the situation. In the hierarchical pattern, hierarchies among central cloud and collaborative gateways are designed.

The *FCE* of the Melle-Longchamp area should deal with three modes:

- **Mode 1:** *Fast action mode*. Due to a critical situation of the transmission network, a simple flow chart logic is used to activate the circuit breakers only. This mode is a fall-back plan of mode 2 as well. The computation consumes a low CPU.
- **Mode 2:** *Normal mode*. The *MPC* solver base the computation on a cost function to give the optimal use of all levers (wind-farms modulation, batteries, and circuit breakers) on a 60 seconds horizon. If no solution is found in the allocated time slot, it switches to mode 1. The computation consumes medium CPU.
- **Mode 3:** *Enhanced forecasting mode*. A more sophisticated *MPC* provides data-driven forecasts which enhance the predictions on generation. The computation consumes high CPU.

It is worth mentioning that the system's situation is shown on the operators' dashboards in all three modes. Each of the modes has an occurrence probability. The probabilities that come from the *RTE* data-driven estimation specify the amount of time each mode is in operation. The probabilities are stipulated as 10% for Mode 1, 60% for mode 2, and 30% for mode 3.

5. The *fork/join* nodes split the sampled data for different sensors and/or actuators sets. These nodes facilitate adaptation in *sense* and/or *actuate* levels by, e.g., specifying the routing probabilities for each brand-new task type heading to sensors and/or actuators.

6. The actuation plan is implemented by actuators to achieve common goals. In our smart grid system, the dashboard components receive the data every second, but other actuation types perform in an event-based manner. The loop is complete when the actuation on the environment is perceived again by sensors.

Table 1. *IAS-QN* task types and CPU times for the Melle-Longchamp case.

| IAS-QN Layer | Service Center | Task Type Name | Mean Service time (milliseconds) | | |
|--------------------------|-------------------------------------|---------------------|----------------------------------|------------------------|--------------------------|
| | | | Centralized (RTE) | Distributed (Gateways) | Hierarchical (Fog-Cloud) |
| IoT Elements (sensors) | Current/voltage | CVSense | | 200 | |
| | Position Relays | PRSense | | 500 | |
| Network | Sense to Gateways (td+pd) | CVSense and PRSense | | 1 | |
| | Gateways to Controllers (td+pd) | CVSense and PRSense | 400 | - | 1200 |
| Processing | ACE | Mode1, 2, 3 | 10 | 25 | 1 |
| | | Mode1 | 15 | 35 | 1 |
| | FCE | Mode 2 | 200 | 440 | 1 |
| | | Mode 3 | 800 | 1760 | 1 |
| Network | Control to Dashboard (td+pd) | DashboardActuate | | 150 | |
| | Control to Circuit Breakers (td+pd) | BreakerActuate | | 150 | |
| | Control to Batteries (td+pd) | Battery Actuate | | 300 | |
| | Control to Windmills (td+pd) | WindmillActuate | | 300 | |
| IoT Elements (actuators) | Dashboards | DashboardActuate | | 1 | |
| | Circuit Breakers | BreakerActuate | | 100 | |
| | Batteries | Battery Actuate | | 1000 | |
| | Windmills | WindmillActuate | | 20000 | |

4.5 Simulation

The *IAS-QN* is modeled and simulated in *JMT 1.0.5* [24]. We ran all the experiments on a *Corei7 2.7GHz* computer with *16Gb* of RAM memory under Windows 10 pro *64-bits*. While flowing through the *IAS-QN*, each task takes a certain amount of service (CPU) demand on each visited station. The CPU depends on the job class associated with the tasks. Table 1 shows mean service time on each *IAS-QN* component and layer. Workload intensities that are the entry rate of job classes to *IAS-QN*, must be specified as well. In our application scenarios, the workloads associated with transducers and relays are 1 second and event-based, respectively. As already mentioned, the architectural self-adaptation within our process is addressed by mode adaptation. Mode adaptation relies on class-switch routing probabilities, i.e., the probability of monitoring tasks routed to functional controllers. In this study, we are mainly concerned with *mean system response time*, which is the mean time spent from sampling to the time that actuation ends.

We tested three architectural patterns (see Figure 7) and their transition to assess their impact on the system's performance. Figure 9 shows the mean response time associated with the smart transmission network managed by our self-adaptive pattern transition approach. We considered 21 different scenarios resulted from the architectural patterns' combinations for handling the three modes. 18 out of 27 scenarios address the transition between patterns (i.e., run-time adaptation),

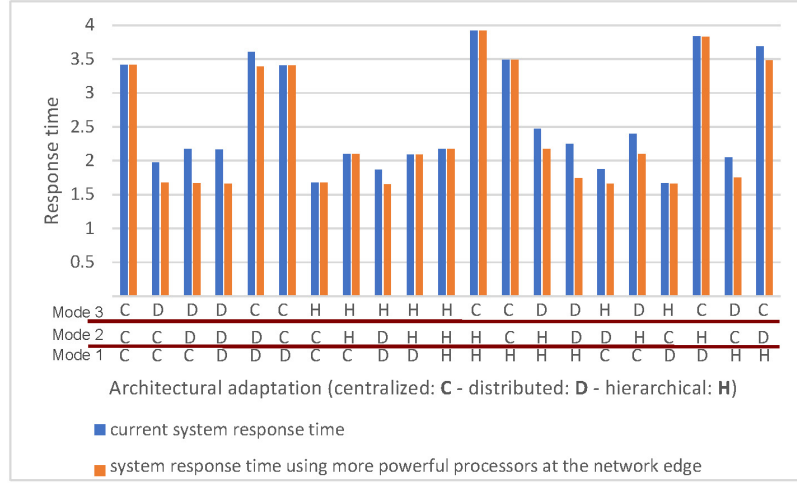


Fig. 9. The experimental results: response time (seconds).

each being in charge of managing a specific computation mode of the Melle-Longchamp *IoT* architecture.

4.6 Architectural Design Decisions

Experimental results on system response time (blue bars of Figure 9) show that managing the fast action mode (*mode 1*) and the normal mode (*mode 2*) with the centralized, and the enhanced forecasting mode (*mode 3*) with the hierarchical architecture minimizes system response time (1.66 seconds). Furthermore, adapting the architecture from distributed (for *mode 1*) to centralized (for *mode 2*), and hierarchical (for *mode 3*) provides the same optimal response time (1.66 seconds).

In several *IoT* systems, the architectural adaptation can take place only on sensors and actuators levels. This might happen, e.g., due to the restrictions on algorithm distribution, hardware resources availability, or middleware design. Thus, if we ignore pattern transition for our smart grid system, we see that, compared with only distributed or only hierarchical, managing the situation with only the centralized pattern increases the delay by 58%. Apart from the fact that the performance depends on how much the processing and storage components are pushed to the edge in a decentralized way, other *QoS* consideration may entirely change the story. If we prioritize, e.g., the *fault-tolerance* of the system, using a centralized pattern causes a single point of failure. Thus, a hierarchical architecture can guarantee the fault-tolerance [22] since if one fog node fails, the *IoT* system can shift the computation to another fog to avoid the single point of failure.

Furthermore, we tested using the more powerful computing resource (i.e., the same as RTE central control element) distributed at the edge of the network (i.e., gateways). The corresponding result is shown as the orange bars in Figure 9. The results show an improvement in response time in all pattern transitions where the distributed pattern is involved. This upgrade highlights the *only distributed* pattern as the optimal solution, by an 11% response time improvement over the previous optimal solution. Thus, taking advantage of our *IAS-QN*, we proposed the RTE company to *i*) re-design their software architecture adaptation policy to manage their *mode 2* with centralized

and *mode 3* with hierarchical, while choosing among centralized and distributed patterns for mode 1; *ii*) push the powerful *FCE* to the edge of the network in a distributed collaborative way.

Lessons Learned. The modeling and development of the Melle-Longcham area smart grid system are still ongoing. However, we learned that adopting a run-time architectural adaptation mechanism is crucial, specially to set the propositions to enhance the performance of the system. We believe that applying *IAS* could bring various benefits to *IoT* systems. We notably learned that Internet of Things architectures require containing the mechanisms to determine the architectural adaption based on their *QoS* satisfaction level. In our use-case, the architectural adaptation performed by changing the computational components' combination to satisfy the performance requirements. However, the adaptation can also take place in sensing, network, and actuating components. The adaptation can be considered internal to the system. The autonomic control element can analyze the situation of functional control elements in run-time, and plan for specific architecture variant adaptation. Architecture variant determines multiple functional deployment types as patterns. In our use-case, this process was executed by class-switch in *QNs*, which enabled a run-time pattern adaptation for performance improvements.

5 Conclusion

This paper presents a conceptual framework for *IoT* Architectural Self-adaptation (*IAS*). The approach facilitates architectural adaptation by correlating it with autonomic and functional control elements. The method is further modeled within Queueing Networks to provide architecture-based performance assessments. We took advantage of the *IAS* framework to design and improve the architecture of *RTE Company*'s transmission network, located in the Melle-Longchamp area (France). By modeling the interaction among autonomic and functional control elements, we designed and further improved a set of *IAS-QN* models that take advantage of *MAPE-K* approach for desirable run-time adaptation. We observed that a proper architecture could keep the response time in a level that is compliant with real-time requirements. We also noticed that some architecture patterns and their switch provide similar response times. Thus in future work, we will consider other complementary criteria (e.g., resiliency) to make architectural design-decisions. We will also apply our approach to test other performance indices. Another improvement that can be performed in future work is formalizing both the run-time pattern selection process and sampling rate settings.

References

1. Danny Weyns. Software engineering of self-adaptive systems: an organised tour and future challenges. *Chapter in Handbook of Software Engineering*, 2017.
2. ISO/IEC/IEEE. Iso/iec/ieee 42010, systems and software engineering – architecture description. 2011.
3. Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaella Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M Göschka. On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, pages 76–107. Springer, 2013.
4. Radu Calinescu, Simos Gerasimou, and Alec Banks. Self-adaptive software with decentralised control loops. In *International Conference on Fundamental Approaches To Software Engineering*, pages 235–251. Springer, 2015.
5. Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaella Mirandola, and Giordano Tamburrelli. Dynamic qos management and optimization in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2010.

6. Gueyoung Jung, Kaustubh R Joshi, Matti A Hiltunen, Richard D Schlichting, and Calton Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *2008 International Conference on Autonomic Computing*, pages 23–32. IEEE, 2008.
7. Edith Zavala, Xavier Franch, Jordi Marco, and Christian Berger. Hafloop: An architecture for supporting highly adaptive feedback loops in self-adaptive systems. *Future Generation Computer Systems*, 105:607–630, 2020.
8. Betty HC Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *International Conference on Model Driven Engineering Languages and Systems*, pages 468–483. Springer, 2009.
9. Stepan Shevtsov and Danny Weyns. Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 229–241, 2016.
10. Arjun P Athreya, Bruce DeBruhl, and Patrick Tague. Designing for self-configuration and self-adaptation in the internet of things. In *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 585–592. IEEE, 2013.
11. Muhammad Usman Iftikhar, Gowri Sankar Ramachandran, Pablo Bollansée, Danny Weyns, and Danny Hughes. Deltaiot: A self-adaptive internet of things exemplar. In *2017 IEEE/ACM SEAMS*, pages 76–82. IEEE, 2017.
12. Danny Weyns, Gowri Sankar Ramachandran, and Ritesh Kumar Singh. Self-managing internet of things. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 67–84. Springer, 2018.
13. David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.
14. Henry Muccini, Romina Spalazzese, Mahyar T Moghaddam, and Mohammad Sharaf. Self-adaptive iot architectures: An emergency handling case study. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, pages 1–6, 2018.
15. David Garlan, Bradley Schmerl, and Shang-Wen Cheng. Software architecture-based self-adaptation. In *Autonomic computing and networking*, pages 31–55. Springer, 2009.
16. Danny Weyns, M Usman Iftikhar, Danny Hughes, and Nelson Matthys. Applying architecture-based adaptation to automate the management of internet-of-things. In *European Conference on Software Architecture*, pages 49–67. Springer, 2018.
17. Eric Rutten, Nicolas Marchand, and Daniel Simon. Feedback control as mape-k loop in autonomic computing. In *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 349–373. Springer, 2017.
18. Philippe Lalanda, Julie A McCann, and Ada Diaconescu. *Autonomic computing: principles, design and implementation*. Springer Science & Business Media, 2013.
19. Claudio Arbib, Davide Arcelli, Julie Dugdale, Mahyar Moghaddam, and Henry Muccini. Real-time emergency response through performant iot architectures. In *International Conference on Information Systems for Crisis Response and Management*. ISCRAM, 2019.
20. Henry Muccini and Mahyar Tourchi Moghaddam. Iot architectural styles. In *European Conference on Software Architecture*, pages 68–85. Springer, 2018.
21. Julie Dugdale, Mahyar Tourchi Moghaddam, and Henry Muccini. Human behaviour centered design: Developing a software system for cultural heritage. In *International Conference on Software Engineering. ICSE-SEIS'2020*, pages 85–94. ACM, 2020.
22. Mahyar Tourchi Moghaddam and Henry Muccini. Fault-tolerant iot. In *International Workshop on Software Engineering for Resilient Systems*, pages 67–84. Springer, 2019.
23. Olaru S. Maeght J. Straub, C. and P. Panciatici. Zonal congestion management mixing large battery storage systems and generation curtailment. In *IEEE Conference on Control Technology and Applications (CCTA)*, pages 988–995. IEEE, 2018.
24. Casale G. Bertoli, M. and G. Serazzi. Jmt: performance engineering tools for system modeling. In *ACM SIGMETRICS Performance Evaluation Review*, pages 10–15. ACM, 2009.