Hierarchical Control for Self-adaptive IoT Systems
A Constraint Programming-Based Adaptation Approach

Mahyar T. Moghaddam
University of Southern Denmark
mtmo@mmmi.sdu.dk

Eric Rutten
INRIA Grenoble
eric.rutten@inria.fr

Guillaume Giraud
RTE Paris
guillaume-np.giraud@rte-france.com

Abstract

The self-adaptation control of Internet of Things (IoT) systems ought to tackle uncertainties in the dynamic environment (application level), as well as the dynamic computation infrastructure (architecture level). While the control of those two levels is generally separated, they should coordinate to guarantee functionality and quality. This paper proposes a conceptual model for the separation of concerns in controlling the environment and infrastructure events. The approach is applied on a real case: Melle-Longchamp area's smart power transmission network (in France). A hierarchical architecture with a control mechanism formalized with constraint programming (CP) is modeled. The control system assesses the reconfigurations that enhance the quality of service (QoS) while considering the internal and external limitations. The CP considers the desired application level control modes and assesses their feasibility by computing the response time and availability using a Netflow algorithm. The outcomes of this research supported design decisions and provided architectural reconfiguration solutions to the French Power Transmission Company (RTE).

keywords. Software architecture, Self-adaptation, Constraint programming, Performance, Smart grid.

1. Introduction

The Internet of Things (IoT) includes a large number of devices across various domains that are interconnected to exchange data and provide services. Due to distribution, heterogeneity, context-dependability, and diversity of languages and protocols, the internal and external risks impact the IoT systems' functionality and quality. In response to these challenges, self-adaptation systems equip IoT software systems with capabilities to cope with contextual and internal dynamic risks. Self-adaptation is generally performed by control elements that interact with system components and the environment to guarantee functionality and QoS. The adaptation control is typically performed by an adaptation manager that comprises the application logic and supervises the managed system [1]. Making an adaptation decision based on the mix of all adaptation possibilities in the *application* and *infrastructure* levels may not be traceable since the manager should cover a huge search space to find an adaptation decision. We thus believe that separating those two adaptation levels, concentrating on each level, and providing a consensus (or coordination mechanism) between the two levels could solve the issues.

In previous work [2], we used queuing networks (QNs) approach to model various control loops in charge of architectural and application adaptations. However, some questions should still be answered: *i)* can the previous approaches separate the application and architecture level concerns while addressing their coordination to ease the adaptation decision-making? *ii)* do previous methods which decide the composition of computing elements in a probabilistic way optimize the QoS? *iii)* how could the trade-off among several non-functional requirements be considered to create additional decision-making value? This paper introduces novel methods to answer those questions.

The separation of concerns concept presented in this paper is formalized with a constraint programming (CP) approach. The CP first models the desired control functions and specifies their priorities based on the environmental risks. The feasibility of those control mechanisms is further evaluated by a CP in charge of assessing the quality of service associated with the infrastructure and architectures. The CP formulates and solves a linearized, time-indexed flow problem on a network representing feasible packet transmission at a suitable frequency. Moreover, the proposed model could be easily evolved to integrate new application or architecture characteristics to allow new QoS-related constraints to be developed and added.

We apply our approach to a real smart grid application: the Melle-Longchamp area (France).

URI: https://hdl.handle.net/10125/80258
978-0-9981331-5-7

The main problem is dealing with overcurrent in the transmission lines due to the excess of energy generation in windfarms, which creates danger for people and goods situated under the transmission lines. Overloading in the lines happens because environmental dynamics should be managed at application and architecture levels by suitable actuation. We design the self-adaptive software architecture of the Melle-Longchamp case with a CP core to deal with the issues mentioned above. We evaluate a set of scenarios that consider various application control modes and architectural reconfigurations to establish a reliable hierarchical control mechanism for RTE. The paper makes the following main contributions:

- Presenting a conceptual model that, while separating the concerns on environmental and architecture levels adaptation, addresses their coordination.
- Applying the approach concretely on a real smart grid system to establish effective operation;
- Designing and evaluating a constraint programming model with a structure enforcing the conceptual separation of concerns. The CP model hierarchically controls various system and environment aspects to enhance the adaptation decision-making quality.

The paper is organized as follows. We performed a literature review presented in Section 2. The separation of adaptation concerns is presented in Section 3. We specify the problem in Section 4 and formulate it in Section 5. The experimental results and lessons learned are discussed in Section 6, and the conclusions are drawn in Section 7.

2. Literature Review

Predicting the uncertainties an IoT system could face during its operation is challenging but necessary. Such predictions at design-time (often outcomes of modeling) input the knowledge-base of the system at run-time. Several configurations are generally modeled to be composed and selected based on dynamic goals and situations. In this way, software architectures could be designed based on expectations and further adapted based on run-time dynamics. Adaptive systems could specify strategies or generate different architectural instantiations as models to find the best way of adapting the structure and behavior of the system to changes [3]. Predefined strategies require design effort but impose a low computation overhead, while run-time model generations could explore optimal solutions with higher computation power needed. While using architectural adaptation provides benefits, the architecture's control mechanism that manages such adaptation could impact the efficiency of adaptation. Optimization algorithms could play the role of the adaptation decision-maker.

Self-optimized systems aim to adapt to changes that may occur in their operational contexts, environments, and system requirements in an optimized manner [4]. Self-optimization through runtime adaptation guarantees not only the functionality but also an optimal trade-off among QoS requirements. Constraint programming (CP) is used to act as architectural reconfiguration decision-maker [5]. Mathematical models could represent different dimensions and characteristics of a system. The variations of resource consumption, functions' migration, network allocation, and reconfiguration cost could be set to optimize the functionality and QoS. The objective functions support the selection of the most suitable configuration regarding specific QoS dimensions. More specific for IoT, CP techniques could find a suitable configuration considering the distributed resources impacted by contextual changes [6]. Using this decision method could find a compromise between contradictory systems and context dimensions. On the system side, managing edge/fog/cloud based on CP could balance cost and revenues while meeting the service and system constraints [7]. Modeling major IoT systems with the distributed collaborative cloud may impose challenges regarding scalability. At the fog level, CP could facilitate the development of generic service placement [8]. Using CP for service placement can enhance deployment constraints and objectives and provide a trade-off between resolution times and solutions quality.

Optimization algorithms are used in the smart grid domain but mostly in optimization of power flow that deals with environmental concerns only (not QoS) [9]. Some studies addressed QoS solutions to enhance the performance of communication between the power supplier and power customers. [10] derives the smart grid QoS requirements via an optimization problem that maximizes the total revenue and then proposes a routing algorithm for the requirement of high-speed routing.

The approach presented in this paper builds on top of the mentioned literature to present a conceptual approach for application and architectural adaptation by designing a CP that manages the adaptation hierarchically. Our approach enables the IoT system to monitor its own situation and the environment to reflect it into the system's architecture, realize opportunities for QoS improvements, select a set of priorities and actions, and execute changes within a feedback control loop.

3. Separation of Adaptation Concerns

3.1. Dynamics

Environment and infrastructure levels. Self-adaptive systems generally deal with dynamic environment as well as dynamic computation infrastructure (Figure 1.a). Traditionally, those systems are represented by managing, and managed systems that get input from the environment [1] (Figure 1.b). The managed system comprises the application logic that provides the systems domain functionality. The managing system supervises the managed system and comprises the adaptation logic. The environment is the real world by which the software system interacts [11]. The environment includes both physical and virtual elements, that the system might not directly control their functionality. We emphasize that a self-adaptive system should be able to sense both contextual and system events to act through physical and architecture adaptation actuators.

Two dimensions of self-adaptation. In this work, we target a generic architecture for a decomposition between the two adaptation levels. As shown in Figure 1.a, it is possible to consider the dynamicity in different manners. Static infrastructures in a static environment, such as traditional large numeric computations in High-Performance Computing (HPC), concerned most computing systems until the 80s or 90s. Dynamic environment with static computation infrastructure was the case of, e.g., traditional hard real-time systems where their very static implementation involved no pointers for reasons of strict predictability. This approach is used in very dynamic environments like physical processes control (e.g., engines, signal processing). Dynamic computation infrastructure that does not consider changes in the environment is the case of, e.g., off-line applications like simulation, modern large numeric computations in HPC with dynamic resource management. This dimension includes systems that deal with dynamicity from fault tolerance and ”resilience or loss of resources through stealing” by more priority applications or jobs. Dynamic computation infrastructure, in coordination with the dynamic environment management, is the more general case where IoT/CPS belongs. In this space, both dimensions have to be managed, but keeping a separation of concerns with adequate coordination can help to face the complexity of the design.

3.2. Coordination architecture

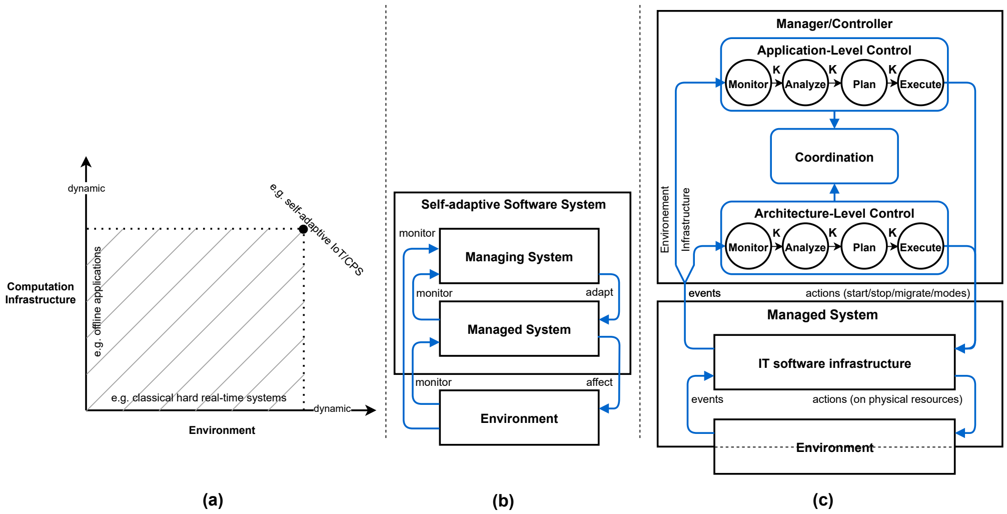
We propose the separation of concerns in controlling the environment and infrastructure events. As shown in Figure 1.c, the environment includes non-controllable hardware, software, and physical space context that interact with the IoT physical resources. The sensors frequently retrieve raw data of events, and actuators receive periodic action commands to affect the environment. The mentioned data transmission is continuous or event-based since the environment is not under full control of the software system, and the dynamics of the environment should be tackled.

In our approach, the managed system represents the composition of IoT resources in the form of software architecture models. An architecture, influenced by the environment, guarantees specific functionalities and QoS. The architectures could therefore adapt to keep the desired functional and QoS according to objective/policy. In this paper, modeling and further simulating different architectures could support the design decisions and tackle environmental and system uncertainties. As shown in Figure 1.c, the manager gets inputs from generated events to analyze and plan for the system functionality by application-level control and for the resources placement, migration, and activation by architecture-level control. In other words, the application-level control comprises the adaptation logic that allows the system to perform the intended adaptation within the environment and architecture-level control supports a continuous reconfiguration in sensing, computing, and actuating elements composition.

It is worth mentioning that the two levels of control coordinate to adjust to each others' situations and requirements. For instance, when the application requires specific functionality with QoS, the infrastructure chooses specific architectural placements to satisfy those needs. On the contrary, if the infrastructure faces some constraints regarding the availability or performance of the resources, the application chooses graceful degradation. The coordination between the two levels can take care of differences of methods and tools used for each of them, e.g., managers can be designed in a simple programmatic approach, using a general-purpose language (C, Java, or Python), or a more specialized language like EAC rules (Event-Condition-Action), or behavioral models such as in Control Theory (continuous or discrete). Other methods are CP, as is done in deployment management in distributed infrastructures [8], or machine learning (e.g., Reinforcement Learning in a feedback loop).

3.3. Architecture Reconfiguration

The above-explained approach could support architecture reconfiguration. Figure 2 shows the generic hierarchical architecture of IoT systems that could be changed to distributed collaborative or centralized if the situation requires. As shown in Figure 1, both the architecture and application levels adopt MAPE-K (Monitor- Analysis- Plan- Execute- Knowledge) [1] control loops. At the application level, the MAPE-K loop affects the environment. At the architecture level,



dynamic
e.g. self-adaptive IoT/CPS
Computation Infrastructure
e.g. offline applications
e.g. classical hard real-time systems
dynamic
Environment
(a)
Self-adaptive Software System
monitor
Managing System
monitor
adapt
Managed System
monitor
affect
Environment
(b)
Manager/Controller
Application-Level Control
Monitor
K
Analyze
K
Plan
K
Execute
Coordination
Architecture-Level Control
Monitor
K
Analyze
K
Plan
K
Execute
Environement
Infrastructure
events
actions (start/stop/migrate/modes)
Managed System
IT software infrastructure
events
actions (on physical resources)
Environment
(c)

Figure 1. a) dynamicity of computation infrastructure and/or environment, b) classical self-adaptation approach, c) the proposed approach on separation of concerns for application and architecture self-adaptation.

it chooses among different instances of the architecture in run-time to act on the system. At the architecture adaptation level, decision-maker (a CP in this paper) acts as the *analysis* and *planning* of the MAPE-K loop. The goal is to automatically compute and plan a new suitable system configuration model from an original one. It could decide about the optimal placement of functions on edge, fog, and cloud nodes.

In the beginning, various possible configurations could be stored in *knowledge* base. The *monitor* component frequently gathers the system's state information, and the potential for the reconfiguration will be assessed by the *analysis* component. The decision maker further *plans* for adaptation by considering the current configuration model. It then produces a set of *executions* that have to be applied to possibly migrate functions to other local or remote control elements. The executor relies on actuators deployed on the edge/fog/cloud to apply the migration actions [12, 13]. According to the run-time conditions and constraints, the mentioned process can be re-executed as many times as required. It is worth mentioning that, since a real IoT system needs to run several functions simultaneously, a level of computation might be optimally performed locally and on another level remotely [14]. The approach presented could help dynamic distribution and placement of function between edge, fog, and cloud. This dynamic migration of software at the edge/fog/cloud highly impacts the functionality and quality of IoT systems.

4. Problem Specification

Built upon [2], the conceptual model presented in the previous section is applied to the RTE company's transmission network in the Melle-Longchamp area (France). The smart grid network includes 35 substations connected by 30 lines. In addition to the power flowing through the network, the area contains wind farms with a total peak production capacity of 300 MW. Such renewable generation is mainly connected to the 20kV distribution grid but impacts the 90kV transmission grid as well. Based on research performed, the RTE company shifted from a traditional PLC-based system to a smart adaptive system with hierarchical decision-making to improve the transmission network's ability to connect more generations without building new power lines. The area's control infrastructure takes advantage of servers on substations, gateways (fog nodes) in a potential collaboration, and a powerful central processor or alternatively cloud.

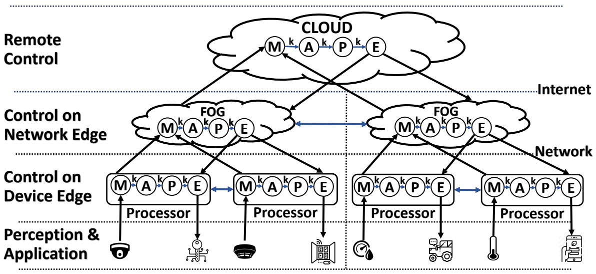


Figure 2. IoT hierarchical architecture.

In the RTE case, we face two main challenges that concern environment and infrastructure dynamics:

1. The potential overload on transmission lines due to unplanned generation from renewable generators, e.g., wind farms, when unexpected strong wind is blowing needs to be managed. This context-related problem may impose safety issues for humans or objects located under the transmission lines.

2. To deal with environment-related issues, the computation resources and functions should be available. In addition to that, the performance should be kept at an acceptable level to let the safety-critical system quickly react in real-time.

4.1. Application Level Control

For the environment related concerns, various solutions could be foreseen, such as limiting the production on wind farms, charging batteries, opening circuit breakers, or a combination of these solutions [15]. Dealing with transmission overload risk necessitates an area-wide control 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 [2]. At this level, an adaptation of modes may be needed to handle the situation. RTE has three control modes to manage the physical environment:

- **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 Model Predictive Control (MPC) solver bases 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.* The system receives forecasts for the next day's generation and weather (wind and sunlight). It also receives the day-ahead studies result indicating the level of risk in the area (alert condition is the higher risk assessment). An enhanced MPC with a 4-hour horizon uses this data on its model to determine if strong variations of generation will occur if they will last for a long time. It also considers the generators' plan to stop during the day (e.g., for maintenance purposes). With all these parameters, it can make smarter decisions with smoother operation. The computation consumes high CPU.

By considering the modes mentioned above, the choice between modes 3 and 2 is triggered by external conditions (application). When New Area Zonal Automatons (NAZA) platform [15] (an application level controller used by RTE) gets all information needed, and the infrastructure has no failure, mode 1 is not generally used. If a risk is detected, mode 3 is preferred. In the absence of risk, mode 2 is preferred to be adopted. It is worth mentioning that the risk level is determined by day-ahead studies based on wind and grid forecasts. It is essential to consider that the architecture level control mechanism presented in the following subsection decides the feasible modes based on the availability and performance requirements and assigns the feasible degraded modes if needed.

4.2. Architecture Level Control

The architecture level control represents the system as a hierarchical software architecture with various levels of control, intending to deal with QoS challenges. Figure 3 shows the global architecture of the area control network. It consists of sensing, network facilities, various processing and storage layers, as well as actuation. Substations' gateways are industrial servers such as *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 the 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 local controllers. Local controllers retrieve and store data from the gateways, perform the computation, and, based on the run-time requirements, either send the data to the cloud or send the orders to the actuators.

As already mentioned, controlling the application adaptation is performed by NAZA, which could run on all fog and cloud nodes. *NAZA* platform principally relies on *RESTful API* to communicate with the gateways. 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 the back office to monitor the system state. The solver implements an MPC or flow chart model (presented above) to optimize a cost function to use levers such as battery set-points and generation limit values. It gets real-time data from the substations 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.

The application level adaptation that chooses and places the NAZA control functions on processing elements is managed by the constraint solving approach presented in Sub-section 5.2. At this level, the

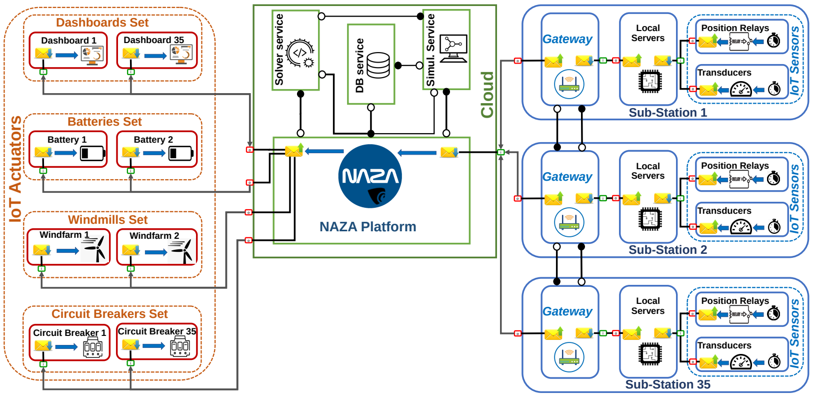


Figure 3. Software architecture of the transmission network.

RTE architecture is assessed based on availability and performance indices, and architectural reconfiguration is proposed. The availability is important since the system shall be available at 99.99% of operation time. Since we have only one control algorithm running at each time in the transmission level, and the algorithm could run on all processing elements over the network, the availability issue could be overcome by dynamically reallocating the material resources (e.g., CPU and bandwidth). The main focus of this work is then performance, since *i)* some types of sensors and actuators have 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. In the RTE case, the computation response time must be kept *less than 2 seconds*.

4.3. Application and Architecture Controls' Coordination

When an environmental risk necessitates adaptation, the architecture controller decides about the reconfigurations and modes that satisfy the system functionality and keeps the system response time within the threshold. To do so, the system should consider various combinations of the processing elements *for each mode*, to assess if the data should be processed at the fog, the central RTE controller, or cloud. Therefore, the feasibility of a preferred mode should be assessed by events on the infrastructure (telecommunication, computing load, missing data) and QoS specifications. If the infrastructure situations make adopting a preferred mode impossible, a degraded mode will be triggered. If no environmental risk is detected, the preference is running mode 2, but if the infrastructure controller sees QoS violation, mode 1 will be adopted. If a risk (such as overload because of strong wind) is detected, the preference is running the system with mode 3. If it violates the QoS, mode 2 will be run, and the last solution will be adopting mode 1. It is important to note that the constraint program-based infrastructure controller assesses the QoS associated with each mode by considering different architectural configurations.

5. Problem Formulation

This section will formulate the control problem in three levels: the application, architecture, and coordination. We keep the CP formulation general to advantage other domains and use-cases as well.

5.1. Application Level Formulation

The application self-adaptation control, which perceives and acts on the environment, could be performed using simple rules, model predictive control, machine learning, and optimization algorithms. In the case presented in this paper, if a strong wind blows, the RTE control system should manage the overload of the transmission lines by activating circuit breakers, batteries, or wind-farm modulations. These are handled by the three modes presented in the previous section. Since the focus of this study goes to architectural reconfiguration, we refer the interested readers to an RTE previous work detailing MPC algorithms [15], which models an overload management system by mixing batteries and renewable generation curtailment. In this paper, we consider the constraints on choosing a mode in the presence and absence of environmental *risks* by adopting a MAPE-K loop: monitoring the environment, detecting and analyzing potential risks, and deciding on mode selection and execution. Therefore, considering B as a Boolean variable specifying the environmental risks, and M= $\{m_1, m_2, \dots, m_n\}$ a set of modes to run on processing element; m_p represents the set of preferred modes at specific time:

$$m_p = \begin{cases} k & \textit{if B=0,} \qquad k \subseteq M \\ z & \textit{if B=1,} \qquad z \subseteq M \end{cases}$$

Section 5.3 explains that the mode selection is exposed to feasibility check by infrastructure level control. The quality tied up with modes and architectures determines either running a preferred mode is possible, or a degraded mode should be adopted.

5.2. Architecture Level Formulation

The construction of the IoT system shown in Figure 3 could be presented as a network (like Figure 4) which includes sensors, communication links, processing elements, and actuators. The corresponding graph consists of nodes and arcs G=(V,A). Nodes correspond to processors, sensors, and actuators. The arcs represent the dependency (network links) between nodes, which are exposed to network constraints. With no loss of generality, arcs are supposed directed. Sensor nodes represent the origins of the network and push packets either frequently or in an event-based manner. The actuator nodes receive the output of the network. For simplicity, all the actuators are connected to an imaginary destination: *node 0*. The computation nodes have the different processing power. This is why placement of computation on fog, central server, or a cloud becomes a crucial aspect for availability and performance. Figure 4 shows examples of reconfiguration possibilities. The upper network shows processing on a central node that could be RTE powerful central server or cloud. This could also represent the case that one of the fog nodes of a substation acts as a central processor. The lower network shows a distributed pattern in which fog nodes are in charge of processing their substation with a potential collaboration and data sharing to obtain a global view. We denote:

$T = \{0, 1, \dots, \tau\}$, set of unit time slots;

n_i = capacity of node i: the maximum nominal amount of packets that i can host to process or service at any time. We also consider a n_c that is capacity of computation node c, when $c \in C \subseteq V$;

y_i^t = state of node $i \in V$ at time $t \in T$, that is, the number of packets being processed in i at t: this number is a known model parameter for $t = 0$;

x_{ij}^t = the number of packets moving from node i to an adjacent node j in $(t, t+1]$;

$b_{ij} = b_{ji}$ = bandwidth (capacity) of the network between node i and node j: this is the maximum amount of packets that, independently from packets queue on node j, can traverse the network in the time unit.

The flow model uses an acyclic digraph D with node set $V \times T$ and arc set

$$E = \{(i,t) \rightarrow (j, t+1) : ij \in A, t \in T\}$$

D models all the feasible transitions (packets moves between adjacent nodes) that can occur in the network in the time horizon T. Transitions are associated with the x-variables defined above, whereas y-variables define the packet congestion at each time. Our objective is to reduce the delay by maximizing the packets arrived at the destination in a minimum time:

$$\max \quad y_0^\tau \tag{1}$$

To find the minimum total time in which packets arrive at the destinations, a *max flow problem* could be solved for different τ looking for the least value that yields (a zero value: if sending packets stops) optimal solution. The method can thus provide the decision-maker with the Pareto-frontier of the conflicting objectives $\min\{\tau\}, \max\{y_d^\tau\}$. Assuming that x and y are integer, the following constraints guarantee the flow of packets in the network and sets the capacities:

$$y_j^t - y_j^{t-1} - \sum_{i:ij \in A} x_{ij}^{t-1} + \sum_{i:ji \in A} x_{ji}^{t-1} = \quad 0 \tag{2}$$

$$j \in V, t \in T, t > 0$$

$$0 \;\leq\; x_{ij}^t + x_{ji}^t \leq \quad b_{ij} \qquad t \in T, ij \in A \tag{3}$$

$$0 \;\leq\; y_i^t \leq \quad n_i \qquad t \in T, i \in V \tag{4}$$

$$S_c^t = \begin{cases} 0; & \textit{if c is missed} \\ n_c; & \textit{otherwise} \end{cases} \qquad t \in T, c \in C \tag{5}$$

Equation (2) is just a flow conservation law: it expresses the packets on node j at time t as the number y_j^{t-1} of packets present at time $t-1$, augmented of those that during interval $(t-1, t]$ transmit to j from another node $i \neq j$, minus those that in the same interval leave node j for another node $i \neq j$. Constraint (3) reflects the limited communication capacity (bandwidth), and constraint (4) puts a computation capacity limitation on nodes. Constraint (5) is designed for the specific case of assuring resiliency if a computation node is missed because of an attack or fault. Thus, if a computation node is missed, its temporal capacity (S) goes to zero, and the situation of the network will be assessed in the absence of that node. The linear structure of the presented model allows its solution with a large number of variables to improve model granularity. More importantly, it can also help approximate the non-linearities of arc capacities. In fact, c_{ij} constant in (3) fails to model packets congestion. A more accurate model of congestion requires arc capacity to be a concave decreasing function of node occupancy. Linearizing this function is quite standard [16] in applications: we consider the three-pieces linearization in our simulations.

5.3. Application and Architecture Coordination Formulation

As already mentioned, the choice of modes is influenced by the preference as well as the QoS. If the set of modes $M = \{m_1, m_2, \dots, m_n\}$ (including k associated to non-risky and z associated to the risky environment) is arranged in decreasing order of preferences, let $H \in \mathbb{R}$ represent the quality threshold and let Q denote the quality function (dynamic based on architectural reconfiguration), then the objective of the optimization function, O, can be defined as a piecewise function, where the objective is to find the value of $i \ni$:

$$O(i) = \begin{cases} m_i; & \text{if } Q(m_i) < H \\ O(i+1); & \text{otherwise} \end{cases}$$

The optimization function aims at finding the highest priority mode in which the associated quality remains under the threshold. We apply our CP-based control approach to the RTE case through the experimentation performed in the following section.

6. Experimentation

We next report the outcome of simulations using the optimization model under various conditions and configurations. We computed the minimum time required of N packets (based on the scenario) to flow through the network towards actuators in all tests. We performed 12 simulations based on architecture patterns and computation modes formulated by CP and 9 simulations on JMT [17]. This allowed us to compare our approach with queuing networks application that is already presented in the literature. To reduce computation time in CP simulations, we computed the optimal response time by logarithmic search, which imposes many simulation iterations. In the simulations, we considered 35 packets per second are sent from the 35 current/voltage sensors located in RTE substations and few packets from position relays in specific events. The code for CP simulation was written in OPL language, and problems were solved by CPLEX version 12.8.0. 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 network (Figures 4), each packet takes a certain amount of service (CPU) demand on each visited node. We set the model parameters using the real data provided by the RTE company. Table 1 shows the mean service time on each component and layer. The table shows the service time for various architectural reconfiguration scenarios and the CPU needed for the three RTE operating modes. The network delays are the sum of transmission and propagation delays. Our CP model proposes the best reconfiguration based on requirements imposed due to environmental changes. Thus, the architectural self-adaptation within our process is initiated by mode adaptation and completed by computation placement on different processors. In this study, we are mainly concerned with *mean computation response time*, which is the mean time spent for computation. We tested four architectures with different computation placements for each mode and situation imposed by the environment. The possible architectural configurations are shown in Figure 4, where the upper network corresponds to central processing on either central RTE processor or cloud, the middle network shows processing on distributed fog, and the lower network shows distributed processing on several collaborative fog nodes on which a distributed version of the control algorithm is running. It is worth mentioning that collaborative fog architecture requires a distributed MPC that RTE has not yet finalized. We simulate that architecture as well to anticipate its potential advantages to RTE managers.

Table 1. Service times for the RTE system.

Layer	Service Center		Mean Service Time (milliseconds)			
			Centralized RTE	Distributed Fog	Collaborative Fog	Hierarchical Fog-Cloud
IoT Eelements (sensors)	Current/voltage		200			
	Position Relays		500			
Network	Sense to Gateways (td+pd)					
	Gateways to Controllers (td+pd)		400		200	1200
Processing	Controllers	Mode 1	15	35	10	
		Mode 2	200	440	150	
		Mode 3	800	1760	600	
Network	Control to Dashboard (td+pd)		150			
	Control to Circuit Breakers (td+pd)		150			
	Control to Batteries (td+pd)		300			
	Control to Windmills (td+pd)		300			
IoT Elements (actuators)	Dashboards					
	Circuit Breakers		100			
	Batteries		1000			
	Windmills		20000			

Simulation 1: ***Mode feasibility in environmental risk situations.*** In the first simulation that intends to deal with the environment, we assess the mode adaptation and architecture reconfiguration based on infrastructural constraints, environmental risk presentation, and mode preferences. Based on Section 4.3, in case of no risk, RTE prefers mode 2. Using the Netflow algorithm presented in this paper for mode 2, all architectural configurations keep the response time under the required threshold of 2 seconds (See Table 2). However, the MPC placement associated with mode 2 in collaborative fog provided the minimum response time. If the response time threshold requirement of RTE changes to 1 second, a transition to mode 1 and an architectural reconfiguration to distributed or collaborative fog will be required. If an environmental risk is detected, the preferred mode 3 could be adopted but only if architectural reconfiguration to collaborative fog or hierarchical fog-cloud styles occurs. If, e.g., using a centralized style is strictly required, the computation component should run on degraded mode 2.

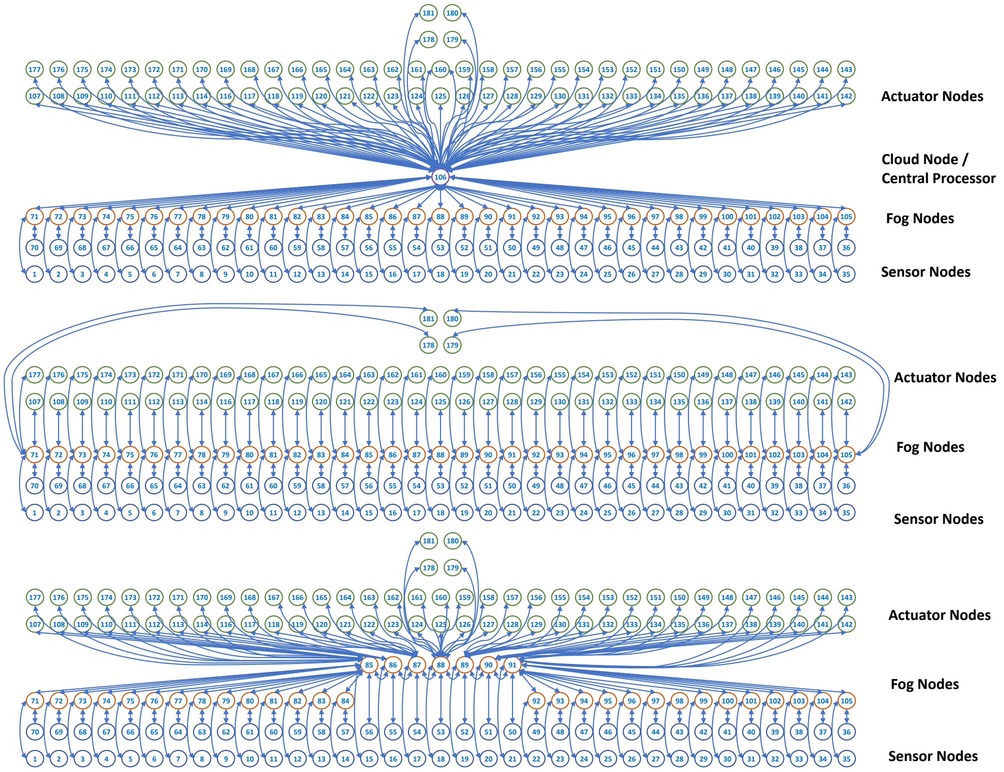


Figure 4. Network associated to Melle area: hierarchical (and centralized not considering fog nodes), distributed, and collaborative.

Table 2. Response time for different modes and configurations (seconds).

	Mode 1	Mode 2	Mode 3
Centralized RTE	1.1	1.4	2.35
Distributed Fog	0.65	1.1	2.3
Collaborative Fog	0.95	1	1.9
Hierarchical Fog-Cloud	1.75	1.75	1.75

Simulation 2: ***Resiliency.*** This case which concerns infrastructure evolution, assumes that some computation nodes are missed due to either fault or cyber-attack. In this case, the centralized styles cause a single point of failure if the central computation component is missed. In collaborative fog configuration (Figure 4, the network on the bottom), we assume that nodes 90 and 91 are missed, and we recalculate the performance. We see that, while the availability requirement is satisfied, the queue of packets increases the response time to 1.15, 1.3, and 2.25 seconds for mode 1, mode 2, and mode 3, respectively. We observe that this situation leads to the performance requirement violation in mode 3.

Simulation 3: ***Comparison with Queuing Networks (QNs).*** We model the exact same system with different architecture configurations with QNs, using a probabilistic routing strategy. We used JMT [17] to model and simulate the QNs. The results showed that our proposed Netflow constraint program reduces the response time (compared with QNs) in all modes and architectural configurations: in centralized style by 22%, 13%, and 6%; in distributed fog style by 52%, 29%, and 13%; in collaborative fog style by 45%, 33%, and 18%; and in hierarchical fog-cloud style by 19%, 19%, and 19%, for modes 1, 2, and 3 respectively. These results could be due to avoiding packets' congestion the Netflow algorithm provides.

Lessons Learned. The work on modeling and development of the Melle-Longchamp area smart grid system is still ongoing. Our proposed approach and experimentation results supported RTE company to set their propositions towards a higher quality smart grid system. We learned that, while the environmental and infrastructural adaptations are performed by different teams, they need to interact, coordinate, and continuously input each other to satisfy functionality and quality. We showed that the coordination of the two levels of control is crucial to overcome the external and internal risks. We learned that constraint solving algorithms could be suitable decision-makers to optimize the intended functionality and QoS. We showed that architectural self-adaptation using CP could provide optimal reconfiguration and migration of software on processing elements. We observed that using the proposed CP is compliant with real-time applications since its computation requires 0.86 seconds (presolve included) in the worst case. Our simulations that are set based on real data support design phases by giving ideas of various scenarios. This could be helpful since, even if some functions and properties are not yet available, they could contribute to the design methods. Based on the results, we proposed to RTE to design the control algorithms that could be distributed to provide a better performance and reduce the possibility of failure. This research's proposed architecture reconfiguration strategies helped RTE deal with the overcurrent situation while improving QoS.

7. Conclusion

This paper presented an approach to separate the concerns on application and architecture level adaptations while keeping their coordination. We formalized the approach with constraint programming and applied it to a real smart power transmission case. We established a feedback loop that considers the preference on control algorithms and the possible QoS satisfaction that architectural reconfiguration could provide. Our approach proposed optimal architectural adaptations to keep the response time at an acceptable level. As future work, from the point of view of constraints and models, we want to consider more dynamics (e.g. speed and acceleration of variations) in relation with Control Theory [18]. We intend to consider resource consumption in our future scenarios to assess the trade-off between performance and energy consumption. The challenge of computation functions placement on fog nodes is here addressed in a simplified way, but a more elaborated consideration based on run-time adaptation will be discussed.

Acknowledgement

This work was financially supported by European commission through CPS4EU project that has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826276. The JU receives support from the European Union's Horizon 2020 research and innovation program and France, Spain, Hungary, Italy, Germany.

References

[1] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka, "On patterns for decentralized control in self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II*, pp. 76–107, Springer, 2013.

[2] M. T. Moghaddam, E. Rutten, P. Lalanda, and G. Giraud, "Ias: an iot architectural self-adaptation framework," in *European Conference on Software Architecture*, pp. 333–351, 2020.

[3] J. Cámara, D. Garlan, B. Schmerl, and A. Pandey, "Optimal planning for architecture-based self-adaptation via model checking of stochastic games," in *Proceedings of the 30th annual ACM symposium on applied computing*, pp. 428–435, 2015.

[4] H. Khazaei, A. Ghanbari, and M. Litoiu, "Adaptation as a service.," in *CASCON*, pp. 282–288, 2018.

[5] C. Parra, D. Romero, S. Mosser, R. Rouvoy, L. Duchien, and L. Seinturier, "Using constraint-based optimization and variability to support continuous self-adaptation," in *Proc. 27th ACM Symp. on Applied Computing*, 2012.

[6] N. Gamez, D. Romero, L. Fuentes, R. Rouvoy, and L. Duchien, "Constraint-based self-adaptation of wireless sensor networks," in *2nd int. workshop on adaptive services for the future internet and 6th int. workshop on web APIs and service mashups*, 2012.

[7] Z. Al-Shara, F. Alvares, H. Bruneliere, J. Lejeune, C. Prud'Homme, and T. Ledoux, "Come4acloud: An end-to-end framework for autonomic cloud systems," *Future Generation Computer Systems*, vol. 86, pp. 339–354, 2018.

[8] F. A. Salaht, F. Desprez, A. Lebre, C. Prud'Homme, and M. Abderrahim, "Service placement in fog computing using constraint programming," in *2019 IEEE Int. Conf. on Services Computing (SCC)*, 2019.

[9] S. S. Reddy, V. Sandeep, and C.-M. Jung, "Review of stochastic optimization methods for smart grid," *Frontiers in Energy*, vol. 11, no. 2, pp. 197–209, 2017.

[10] H. Li and W. Zhang, "Qos routing in smart grid," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pp. 1–6, IEEE, 2010.

[11] J. Cámara, B. Schmerl, and D. Garlan, "Software architecture and task plan co-adaptation for mobile service robots," in *Proceedings of the IEEE/ACM 15th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*, 2020.

[12] H. Muccini and M. T. Moghaddam, "Iot architectural styles," in *European Conference on Software Architecture*, pp. 68–85, Springer, 2018.

[13] H. Muccini, R. Spalazzese, M. T. Moghaddam, and M. Sharaf, "Self-adaptive iot architectures: An emergency handling case study," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, pp. 1–6, 2018.

[14] M. T. Moghaddam and H. Muccini, "Fault-tolerant iot," in *International Workshop on Software Engineering for Resilient Systems*, pp. 67–84, Springer, 2019.

[15] C. Straub, S. Olaru, J. Maeght, and P. Panciatici, "Zonal congestion management mixing large battery storage systems and generation curtailment," in *2018 IEEE Conf. on Control Technology and Applications (CCTA)*, 2018.

[16] C. Arbib, M. T. Moghaddam, and H. Muccini, "Iot flows: a network flow model application to building evacuation," in *A View of Operations Research Applications in Italy, 2018*, pp. 115–131, Springer, 2019.

[17] M. Bertoli, G. Casale, and G. Serazzi, "Jmt: performance engineering tools for system modeling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 10–15, 2009.

[18] M. Litoiu, M. Shaw, G. Tamura, N. M. Villegas, H. Müller, H. Giese, R. Rouvoy, and E. Rutten, "What Can Control Theory Teach Us About Assurances in Self-Adaptive Software Systems?," in *Software Engineering for Self-Adaptive Systems 3: Assurances*, vol. 9640 of *LNCS*, Springer, May 2017.