The Journal of Supercomputing (2022) 78:3883–3904
https://doi.org/10.1007/s11227-021-04011-z

DAACS : a Decision Approach for Autonomic Computing Systems

Imen Abdennadher[1]

Accepted: 4 August 2021 / Published online: 17 August 2021

Abstract
Systems running in ubiquitous environments are characterized by a context that changes frequently. The adaptation of this kind of systems according to the context is a complex task. Autonomic computing has received a great attention as a solution for this increasing complexity, through an architecture based on the MAPE-K loop. Decisions within the phases of the MAPE-K loop have an important impact on the success of systems adaptation. In the literature, many research activities propose decision approaches and frameworks for autonomic applications adaptation. Nevertheless, there is a lack of guidelines for the adaptation decisions design task. In this work, we propose a Decision Approach for Autonomic Computing Systems, called DAACS, which includes recommendations and steps that should be followed by the autonomic applications designers. DAACS was implemented in a Smart Building case study, and it was evaluated in term of the processing time dedicated for the adaptation decisions.

Keywords Ubiquitous applications · Adaptation · Autonomic computing · MAPE-K · Decision guidelines · Smart building

1 Introduction

The high progress [17] of wireless network and new information technologies leads to the apparition of a new challenging concept: the ubiquitous computing. As introduced by Mark Weiser [27], the ubiquitous computing means that computers are omnipresent and integrated in our everyday life and assist us in our daily tasks in an invisible manner. Ubiquitous environments are characterized by a context that changes frequently. The concept context was defined by Schilit and Teimer [25] as a combination of the localization of the user and the identities and states of his

✉ Imen Abdennadher
imen.abdennadher@redcad.org

[1] ReDCAD, University of Sfax, B.P. 1173, 3038 Sfax, Tunisia

surrounding persons and objects. The context includes relevant aspects of the surrounding physical and computing environments, such as the ambient temperature, the users locations and activities, etc. Context-awareness is an essential characteristic of systems running in ubiquitous environments, often called ubiquitous systems. This kind of systems must be able to discover, react to changes of the environments where they are situated [25] and adapt themselves to changes. Adaptations to changes could be classified into behavioral or architectural according to the adaptation modification that affects either the behavior or the architecture of the application of a considered ubiquitous system [4, 10, 12, 15].

The autonomic computing paradigm was arisen to tackle the complexity of designing systems that cope autonomously with their changing environments [12]. The autonomic computing brings systems with self-management capabilities that include self-configuration, self-optimization, self-healing and self-protection [11]. Making decisions is an important task in the adaptation process of autonomic applications. From one side, a faulty decision automatically leads to a faulty adaptation. From the other side, a suitable or a successful decision participates in making a successful adaptation.

The adaptation decisions receive a great attention by the research community and several research activities propose decision approaches and frameworks. Nevertheless, as far as we know, there is a lack of guidelines to design adaptation decisions. We believe that such guidelines provide a considered assistance to the autonomic applications designers and unify the existing decision approaches.

A review of research activities dealing with adaptation decisions shows that there are three main decision approaches: the Situation-actions decision approach, the Goal-oriented decision approach and the Utility-oriented decision approach [14, 21, 29]. The Situation-actions decision approach needs a definition of a set of decision rules at design-time to specify the system's transition from the current state to the next one. In the Goal-oriented decision approach, the responsibility of calculation of the necessary actions to move from a system's current state to a desired one is given to the system itself [9]. The principle of the Utility-oriented decision approach consists on the calculation of the utility value of each alternative based on a utility function, then the selection of the alternative that has the highest utility value (considered as the most suitable alternative).

In this paper, we provide a study that includes a discussion and recommendations for the choice of the decision approaches. Then, we presented our decision approach that includes guidelines that should be followed by autonomic applications designers to assist them in the adaptation decisions design task. Our approach is implemented through a reusable decision module[1] that assists the autonomic applications developers. Our decision approach is applied in a Smart Building case study to adapt the energetic profile of the building, and the implementation of our decision approach is evaluated by calculating the overhead of the time processing of the adaptation decisions.

[1] https://github.com/imenGithub/DecisionModule.git

2 State of the art

We have studied research activities that deal with decisions for applications adaptation. From this study, we concluded that there are three principle decision approaches: the Situation-actions decision approach, the Goal-oriented decision approach and the Utility-oriented decision approach. In this section, we show some representative research activities for each approach and research activities that combine between decision approaches. Then, we end this section by a discussion and recommendations about the decision approaches.

2.1 Research activities dealing with the situation-actions decision approach

Elmalaki et al. [6] present the design and implementation of an adaptation framework, called CAreDroid, for android context-aware applications. The decision in this framework is based on application-specific rules which ensure the mapping of methods to the context. The alternatives of decision correspond to the set of implementations for the same method which either provide the same functionality with different performances or alternative functionality to the same method. The CAreDroid framework switches between methods' implementations at run-time according to a set of context parameters (Battery state, Connectivity state, Location and Mobility state) and based on the specified rules.

Zhao [29] proposes an approach for the planning process of requirements driven self-adaptation that follows the Situation-actions (or the rule-based) approach. The author confirms that the Situation-actions approach provides an efficient offline planning method. However, it is not able to react neither to unexpected environment changes nor changeable requirements. For this reason, the authors suggest a solution that enhances the Situation-actions approach with a rule generation and a rule evolution process.

2.2 Research activities dealing with the goal-oriented decision approach

Peng et al. [23] present a self-tuning method for software systems, based on goal-oriented reasoning and feedback control theory. A goal model includes hard goals that model requirements with clear satisfaction criteria, and softgoals that are formally evaluated through reasoning. The authors propose a preference-based algorithm which configures the hard goals in order to guide the following architecture reconfiguration. The goal-oriented reasoning considers the goal model as a set of logic constraints. The proposed algorithm initially tries to find a configuration which respects all the constraints related to softgoals. If no configuration is found, the lowest ranked softgoals will be removed from the encoding, so as to find a configuration which respects the remaining softgoals' constraints and so on. The decision in the proposed algorithm is based on the ranks of the softgoals. In addition, preference ranks are also assigned to softgoals to express dynamic

quality tradeoff decisions (to differentiate between the softgoals that have a same rank).

Lei et al. [16] propose an agent-oriented self-adaptive software development method which is based on the Goal-oriented approach. The proposed method uses the original Tropos goal model and extends it with external context conditions and internal event conditions, in order to model the adaptive software requirements.

2.3 Research activities dealing with the utility-oriented decision approach

Moreno et al. [19] propose an architecture-based approach for self-adaptive systems. The authors use probabilistic model checking for adaptation decisions. The proposed approach fits the architectures which are based on explicit control loop such as the MAPE-K loop. The self-adaptation is based on maximizing a utility function. In a first step, the adaptation decisions determine whether an adaptation of the system is required. Then, they select the configuration that provides the highest utility and generate the set of adaptation tactics that must be executed (in the execution phase of the MAPE-K loop).

Pascual et al. [21] suggest an approach for self-adaptive mobile system. The proposed approach consists in automatically generating the application configurations and the reconfiguration plans at run-time. The authors use a genetic algorithm, called DAGAME, which optimizes a utility function in order to decide which architectural configuration provides the best functionality and respects available resources.

2.4 Research activities dealing with a combination of decision approaches

In the work of Klös et al. [13], an extension of the IBM's MAPE-K loop architecture is proposed to allow systems to cope with situations that are not anticipated at design time. The adaptation decision in this work is based on a combination between the Situation-actions and the Goal-oriented approaches. The authors impose a structure on the knowledge which includes, in addition to the abstract system model and the abstract environment model, a global goal model and a set of adaptation rules.

Vrbaski et al. [18] propose a context-aware reasoning approach, called CARGO, which combines Situation-actions and Goal-oriented approaches. The authors use the User Requirements Notation (URN), the international requirements engineering standard that integrates goal-oriented and workflow-based modeling. CARGO uses specific extensions to URN that provide the ability to specify and execute context-aware systems.

Gauvrit et al. [8] propose a framework to bring self-adaptability to service-based distributed applications. The proposed framework, called SAFDIS, enables the dynamic evolution of the service-base architectures since it provides the functionalities of the MAPE-k loop. The implementation of the SAFDIS framework includes two reasoners to make decisions. The first reasoner makes short-term decisions, and it is based on the Situation-actions decision approach. The second reasoner handles long-term decisions, and it is based on the Utility-oriented approach.

2.5 Synthesis and positioning of the decision in the MAPE-K loop

Table 1 summarizes the studied research activities dealing with decisions for applications adaptation. Decisions are needed for both architectural and behavioral adaptations and could be based either on the Situation-actions or the Goal-oriented or the Utility-oriented decision approaches or a combination among these approaches.

As shown in Table 1, most of the research activities provide curative decisions for applications adaptation. Furthermore, the decision implementation could vary from a set of algorithms to a framework or middleware or service for decision.

We have studied the positioning of the decision in the MAPE-K loop. A decision is required in the Analysis phase [7, 8]. In fact, the analyzer specifies either it is necessary to adapt the system to the context changes or not. If an adaptation is needed, the request of change is sent to the planner that decides and selects the most suitable reconfiguration actions [5, 8]. Then, a decision in the Execution phase selects the most suitable concrete actions [8].

2.6 Discussion and decision approaches recommendations

The majority of the existing research activities use either one of the decision approaches that we presented (i.e., the Situation-actions, the Goal-oriented and the Utility-oriented decision approaches) or a combination between these decision approaches [22]. Moreover, there are few research activities that define ad hoc decisions, such as the work of Zimmermann [30]. The ad hoc decision approach is specific to an application and fits well its requirements. Nevertheless, it lacks of reusability since it is too customized to the application that it is proposed for.

In general, as observed from the literature, each decision approach has its advantages and its drawbacks [9]. For this reason, many researchers use a combination between decision approaches in order to benefit from the advantages of each one. The Situation-actions approach has a short duration of decision but it is not able to find necessary reactions to unexpected contextual situations. This problem is resolved by the Goal-oriented and the Utility-oriented approaches. The Goal-oriented approach does not require the specification of the possible situations and their associated adaptation actions, but the decision is taken by the system that generates a rational reasoning according to a specified goal model. Nevertheless, the Goal-oriented approach needs, in general, some additional methods to select between a set of possible alternatives, such as adding a priority between goals or rank preferences [23]. The Utility-oriented approach is considered as a special case or an extension to the Goal-oriented approach [21]. The Utility-oriented approach is based on utility functions that calculate the satisfaction degree of the user and aims to choose the alternative that maximizes the overall utility or satisfaction degree.

The choice of the most suitable decision approach depends on the case study. The Situation-actions approach is efficient for case studies that require offline decisions and when there is not unpredictable changes in the context. However, for the highly dynamic case studies, especially those that face unexpected contextual situations,

Table 1 Synthesis of research activities dealing with decisions for applications adaptation

Research activity	Type of adaptation	Decision approach	Type of decision	Decision implementation	Software part responsible for decision	Decision in the MAPE-K loop
Elmalaki et al. [6]	Behavioral	Situation-actions	Curative	Framework	Adaptation engine	Planning (P)
Zhao et al. [29]	Not mentioned	Situation-actions	Curative	Algorithm	Not mentioned	Planning (P)
Wei et al. [26]	Architectural	Utility-oriented	Curative	Middleware	Decision maker component	Planning (P)
Gauvrit et al. [8]	Architectural	Situation-actions and Utility-oriented	Curative	Algorithms	Decision Maker component	Analysis (A), Planning (P) and Execution(E)
Vrbaski et al. [18]	Not mentioned	Situation-Actions and Goal-oriented	Curative	Not mentioned	Not mentioned	Not mentioned
Lei et al. [16]	Behavioral	Goal-oriented	Curative	Algorithms	Jadex agents	Not mentioned
Peng et al. [23]	Architectural	Goal-oriented	Curative	Algorithms	Goal reasoner	Not mentioned
Moreno et al. [19]	Architectural	Utility-oriented	Preventive	Algorithms	Probabilistic model checker	Analysis (A) and Planning (P)
Pascual et al. [21]	Architectural	Utility-oriented	Curative	Genetic algorithm	Dynamic reconfiguration service (DRS)	Planning (P)
Klös et al. [13]	Behavioral	Situation-actions and Goal-oriented	Curative	Algorithms	Not mentioned	Analysis (A) and Planning (P)
Zarghami et al. [28]	Behavioral	Situation-actions	Curative	Decision service	Decision service	Not mentioned

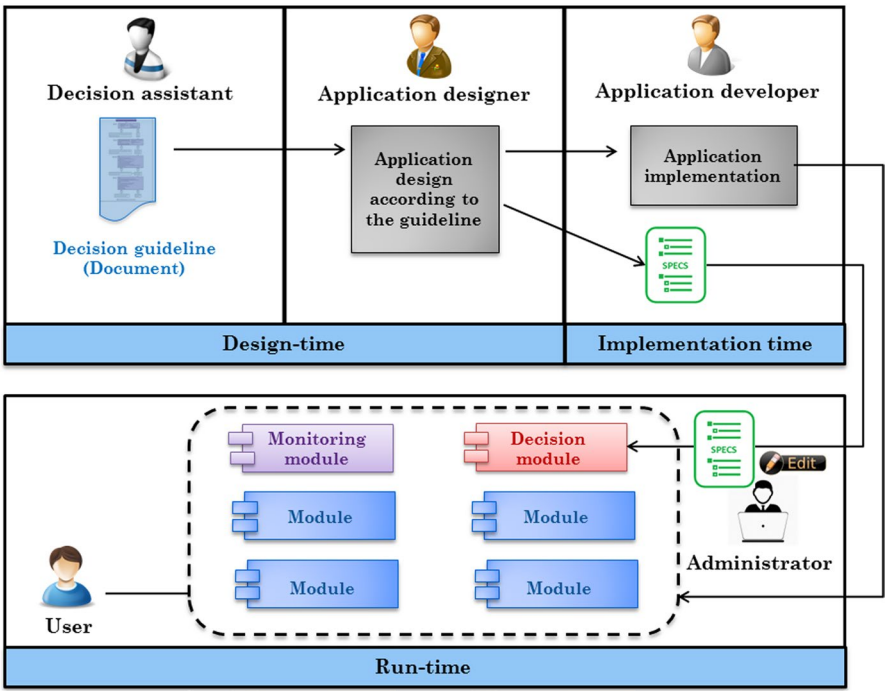

Fig. 1 General view of the decision approach

we recommend the use of either the Goal-oriented or the Utility-oriented approach. Since the Goal-oriented and the Utility-oriented approaches are often computationally expensive (in term of runtime overhead cost), we recommend the use of a combination between these approaches and the Situation-actions approach. In our opinion, decisions are classified into two categories: the decisions that have a time constraint in their duration compared to the duration of the adaptation and the decisions that do not (i.e., they could have enough time without affecting the adaptation). The first category of decisions is often used in critical situations which need rapid reactions, for this reason we recommend the use of Situation-actions approach. For the second category of decisions, we recommend either the Goal-oriented approach or the Utility-approach. In the case where conflicts between goals can appear, we think that it is recommended to use the Utility-oriented decision approach (rather than the Goal-oriented approach) or to use an extended Goal-oriented approach (such as the work of Peng et al. [23] which is based on the goal-reasoning with "dynamic quality tradeoff decisions").

3 The decision approach for autonomic computing systems

Our work consists on the proposition of a decision approach for designing and developing autonomic applications able to detect or predict context and adapt itself to context changes. Figure 1 presents a general view of our decision approach. Three phases of

the life cycle of the application are presented: the design-time, the implementation time and the run-time of the application. The stakeholders who act in the application life cycle are also presented:

- Decision assistant: represents our role, which consists of providing an assistance for the designer of the application (an assistance to design the application adaptation decisions) and an assistance for the application developer by providing a reusable decision module
- *Application designer* is responsible for the design of the autonomic application
- *Application developer* is responsible for the implementation of the autonomic application
- *Administrator* controls the execution of the application at run-time and enhances its functioning when it is possible
- *User* represents the end user of the autonomic application

The decision assistant provides a decision guideline to the application designer to assist him in the design of the decisions that must be taken for the application adaptation. Then, the application designer provides the design of the application according to the decision guideline. The application developer implements the application according to the design and provides a specification that includes the description of information related to the decision.

The decision assistant provides a decision module to the application developer. The decision module facilitates the development of the decisions for the application adaptation. At Run-time, the decision module uses the specification and the notifications of the Monitoring module as input parameters. The notifications of the Monitoring module correspond either to context changes or predictions of context changes. Moreover, the decision module interacts with the application's modules to guide the decisions. The administrator has the possibility to add or modify some details in the specification to enhance the functioning of the application.

3.1 A design guideline for adaptation decisions

The design of the decision as a part of the adaptation process is a difficult and complex task. To overcome this complexity, we propose a decision guideline to assist the designer of autonomic applications. Figure 2 presents the high-level guideline for decision. The designer should follow the steps numbered from 1 to 7 to design a decision for his application adaptation. The arrows in Fig. 2 present input/output relations.

The main role of a decision is to choose among a set of possible alternatives, the most suitable one according to the contextual situation. Each alternative has the type "decision object".

3.1.1 The decision object definition

The first step of the guideline is the definition of the decision object. The latter is the object that the decision is based on, in other words, the set of alternatives are

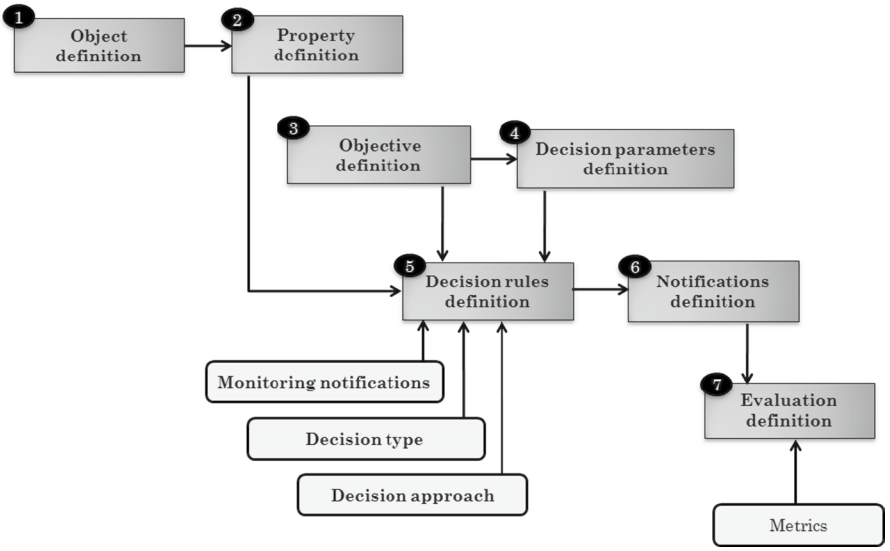

Fig. 2 The design guideline for adaptation decisions

Table 2 Examples of decision objects properties

Decision object	Properties
Architecture of an application	Consumption of resources, dispersion of the components [24], number of reconfigurations [24]
Application	Behavior, structure
Network	Availability, reliability

considered as instances of the decision object. The decision object could be the application's architecture or the application itself or the network, etc.

3.1.2 The decision object property definition

A decision object has a set of properties that characterize it. In this step, the application designer defines the property of the decision object that is directly related to the considered decision. Some examples of decision objects and their properties are presented in Table 2.

3.1.3 The decision objective definition

A decision could have a unique or a set of objectives. In this step, the designer must define the main objective of the decision.

3.1.4 The decision parameters definition

From the already defined objective, the designer defines the decision parameters in the fourth step. For example, if the decision objective is to respect the available resources of machines when deploying the software components of the application, then the decision parameters in this case are the resources of the machines (such as the RAM and the CPU).

3.1.5 The decision rules definition

The property of the decision object as well as the objective and the decision parameters are used to define the decision rules. Before the definition of the decision rules, the designer needs to define the notifications of the monitoring step (of the adaptation process). The monitoring notifications correspond either to the current values of context changes or the predicted values of context changes. Then, the designer defines the type of decision, which could be either curative or preventive. The decision is curative if it is based on the current values of context changes as monitoring notifications. The decision is preventive if it is based on the predicted values of context changes. Once the monitoring notifications and the decision type are specified, the designer should define the decision approach (it could be either the Situation-actions approach or the Goal-oriented approach or the Utility-oriented approach or a combination between these decision approaches).

The definition of the decision rules is highly related to the chosen decision approach. If the Situation-actions approach is chosen, the designer defines a set of rules that determine for each contextual situation the actions that must be executed for the application adaptation. In the case where the Goal-oriented approach is chosen, the designer defines a set of goal policies as decision rules. In the case where the Utility-oriented approach is chosen, the designer specifies a set of utility functions as decision rules.

3.1.6 The decision notifications definition

The notifications of a decision represent the output that will be generated at run-time after making the decision. In the following, we present some examples of decision notifications:

- The ID or the description of the most suitable alternative
- The adaptation actions (that must be applied to set up the most suitable alternative)
- The duration of the decision-making process

3.1.7 The decision evaluation definition

The designer defines the evaluation of the decision according to specified metrics related to the performance and/or the efficiency of the decision results. In the following, we give some propositions of decision evaluation:

- *Evaluation of the efficiency of the decision results* The evaluation of the decision results efficiency could be based on experience from previous application executions. We propose that the application developer should implement a Knowledge module and integrates it with the Decision module and the software components of the application. The Knowledge module should store the pairs <Contextual situation, best alternative> in a database. Within this Knowledge module, the application administrator can compare the result of a current decision with the previous similar decisions stored in the database in order to evaluate its efficiency.
- *Evaluation of the decision duration* The decision duration is the time taken by the decision-making process, which begins with the reading of the set of decision parameters and finish when the most suitable alternative is selected. The decision duration is evaluated by the application administrator according to the criticality degree of the decision. We propose a recommendation to reduce the decision duration that consists on preceding the decision by a preprocessing step. In the preprocessing step, some alternatives are filtered. For example, duplicated alternatives should be filtered and only one of them should be kept for the selection step. Moreover, invalid alternatives or those that do no respect the constraints should be filtered.

4 Smart building case study presentation

The Smart Building case study is presented in details in our previous work [3]. In this paper, we use this case study to evaluate our decision approach. The Smart Building case study includes a set of entities that have different hardware and software capabilities. In this section, we present the hardware elements, then the software elements of this case study.

4.1 Hardware elements of the smart building case study

The building's equipments are classified into seven classes: Controller, Sensor, Actuator, Interface, Smart Meter, Renewable energy source and Energy storage [3, 20].

Figure 3 shows the devices or the hardware elements of our Smart Building case study. The hardware elements of our Smart Building case study are classified as follows:

- Controller of the building
- *Sensors* Presence sensors, Luminosity Sensors, Smart Plug, Thermometer
- *Actuators* Air conditioners, Lamps, Dishwasher
- *Interface* User Agent (responsible for the communication with the user)
- *Smart Meter* Power Smart Meter
- *Renewable energy sources* Photovoltaic Panel
- *Energy storage media* a Battery (to store the surplus of the energy)

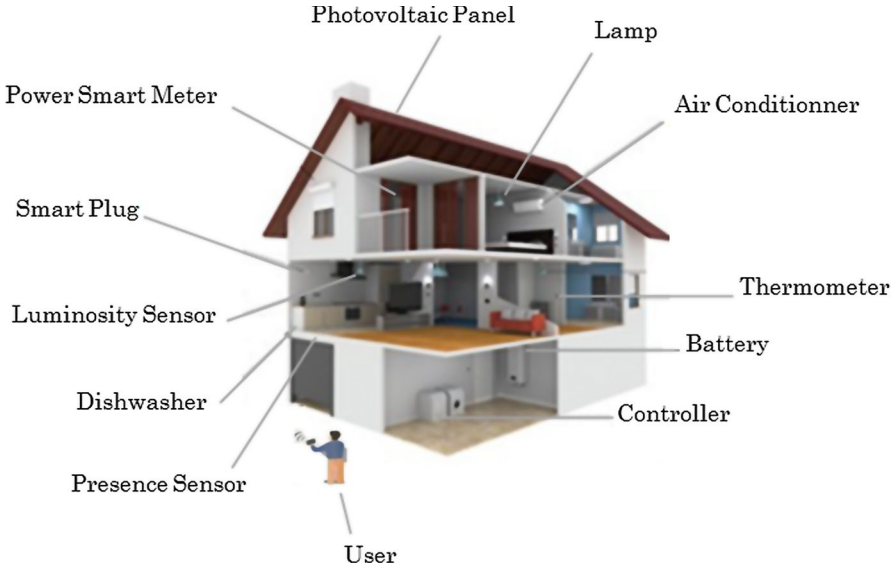

Fig. 3 The smart building case study

A Photovoltaic Panel is responsible for the production of energy to the building through converting the solar energy to an electrical energy. A Battery stores the surplus of the produced electrical energy during low consumption periods and resupplies the building during high consumption periods. A Power Smart Meter measures the electrical energy consumed by the devices of the building (and interacts with a public electrical network in order to inject the surplus of produced energy and pump energy if necessary). A Dishwasher, Air Conditioners and Lamps could adjust their characteristics and functioning modes according to the user's requirements. A Thermometer regularly measures the temperature of the environment. Luminosity sensors generate a signal indicating the light intensity. Presence Sensors detect the presence of proximate objects without any physical contact. Data captured by the sensors is sent to the Controller that evaluates the state of the building and makes assessments.

In the actual implementation of the Smart Building case study, we use six machines that play the roles of the Controller, the Dishwasher, the Power Smart Meter, two Air Conditioners, the User device, respectively. For simplicity reasons, we call each machine by the name of the hardware element. For example, the Power Smart Meter means the machine that we use to play the role of the Power Smart Meter.

In order to manage the devices of the building, we developed an application, called Smart Building Application (SBA), composed of a set of software components. The software components of the SBA are presented in the following section.

4.2 Software elements of the smart building case study

The Smart Building Application (SBA) includes a set of software components[2]. Each hardware element of the Smart Building case study (Controller, Sensors, Actuators, Interfaces, Smart Meter, Energy storage media and Renewable energy source) has its corresponding software element. If the device or the hardware element has enough computing and memory capabilities, its corresponding software component can be deployed on it (For example a software component that manages a Dishwasher could be deployed on this Dishwasher). Otherwise, the component is deployed on the Controller (For example, a software component that manages a Lamp could not be deployed on it, hence it is deployed on the Controller).

The main software component of the SBA is the *Smart Building Manager* deployed on the Controller and responsible for managing the other software components: *Lamp*, *Air conditioner*, *Power Smart Meter*, *Smart Plug*, *Photovoltaic Panel*, *Presence Sensor*, *Luminosity Sensor*, *Thermometer*, *Dishwasher* and *User Agent*. The user can access the SBA by using an interface from his device (smartphone, laptop,etc). The application behind this interface is called *User Agent*.

The software components of the SBA are deployed as follows:

- The *Smart Building Manager* software component, the *Lamp* software components, the *Smart Plug* software components, the *Photovoltaic Panel* software component, the *Presence Sensor* software components, the *Luminosity Sensor* software components and the *Thermometer* software component are deployed in the Controller
- The *Dishwasher* software component is deployed in the Dishwasher
- The *Power Smart Meter* software component is deployed in the Power Smart Meter
- The *Air conditioner* software components are deployed in the Air Conditioners (each *Air conditioner* component is deployed on the Air Conditioner that it manages)
- The *User Agent* software component is deployed in the user device (smartpnone, laptop,etc.)

The *Smart Building Manager* software component centralizes the SBA's data and communicates with the other software components of the application to provide smart and comfortable services to the user. Figure 4 shows the interactions between the software components of the SBA.

- The *Smart Building Manager* software component sends interrogation requests to the other software components to know the values of the characteristics of the devices or the environment (For example, the *Smart Building Manager* could send a request to an *Air Conditioner* software component to know the temperature value of the considered Air Conditioner). The *Smart*

[2] https://github.com/imenGithub/SmartBuilding.git

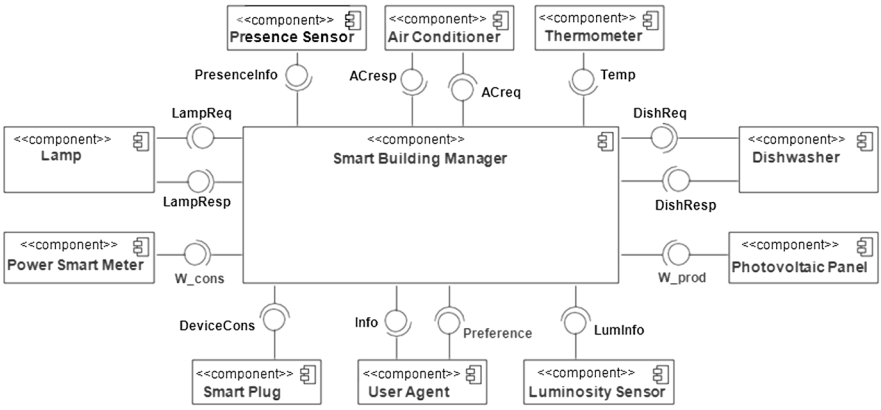

Fig. 4 Interaction between the software components of the Smart Building Application

Building Manager could also send update requests to the *Lamp*, *Air Conditioner* and *Dishwasher* software components to adjust their characteristics (For example, the *Smart Building Manager* could send a request to an *Air Conditioner* software component to decrease the temperature of the considered Air Conditioner).

- The *Photovoltaic Panel* software component sends the values of produced energy (*W_prod* in Fig. 4) to the *Smart Building Manager* software component periodically.
- The *Lamp* software component adjusts its characteristics, i.e., its state (ON/OFF) and its intensity value, according to the building's occupant preferences and the *Smart Building Manager* software component requests.
- The *Air Conditioner* software component adjusts its state (ON/OFF) and its temperature value according to the building's occupant preferences and the *Smart Building Manager* software component requests.
- The *Dishwasher* software component adjusts its state (ON/OFF) and its functioning mode (for example High performance mode or Economical mode) according to the building's occupant preferences and the *Smart Building Manager* software component requests.
- The *Thermometer* software component periodically sends the measured temperature value (*Temp* in Fig. 4) to the *Smart Building Manager* software component.
- The *Presence Sensor* software component periodically sends the rooms' state i.e., empty or not (*PresenceInfo* in Fig. 4) to the *Smart Building Manager* component.
- The *Luminosity Sensor* software component periodically sends the intensity of light (*LumInfo* in Fig. 4) to the *Smart Building Manager* component.
- The *Smart Plug* software component periodically sends the consumption of the device attached to it (*DeviceCons* in Fig. 4) to the *Smart Building Manager* component.

- The *Power Smart Meter* software component periodically sends the value of the energy consumed by the building's devices (*W_cons* in Fig. 4) to the *Smart Building Manager* component.
- The *User Agent* software component sends the occupant's preferences (for example the required intensity of luminosity and the required temperature of the Air Conditioners) to the *Smart Building Manager* component.

5 Experimentation and validation of the decision approach implementation

The goal of this section is to show the applicability of our decision approach and to evaluate its implementation. Our decision approach is first applied in the Smart Building case study to adapt the energetic profile of the building. Then, we present some experiments to evaluate our implementation in term of the processing time dedicated for making the decisions for the building energetic profile adaptation.

5.1 Application of our decision approach for the adaptation of the energetic profile of the smart building

An energetic profile of the building is defined as a collection of the devices behaviors of this building. The decision for the energetic profile adaptation of the building aims to choose the most suitable energetic profile according to the contextual situation related to the electrical energy. In our Smart Building case study, we use two examples of energetic profiles: the "High performance mode energetic profile" and the "Economical mode energetic profile". The High performance mode energetic profile consists on switching on the Dishwasher with its high performance mode as requested by the user, the Air Conditioners with the temperature values requested by the user and the Lamps with the intensity requested by the user. The Economical mode energetic profile consists on switching on the Dishwasher with its economical mode, the Air Conditioners with an increase of the required temperature values by a flexibility value α (α corresponds to the value of sensitivity of the human epidermis to the temperature) and the Lamps with a decrease of the intensity values by a flexibility value β (β corresponds to the value of sensitivity of the human eye to the luminosity).

In fact, in a Smart Building, it is possible to define several Economical mode energetic profiles according to the set of devices functioning in the high performance mode and those functioning in the economical mode. However, for simplicity reasons, we made the choice of using a single Economical mode energetic profile in our case study Smart Building, which consists on switching on all the devices of the building with the economical mode (i.e., the Dishwasher with the economical mode, all the Air Conditioners with the required temperature values $+\alpha$ and all the Lamps with the required intensity values $-\beta$).

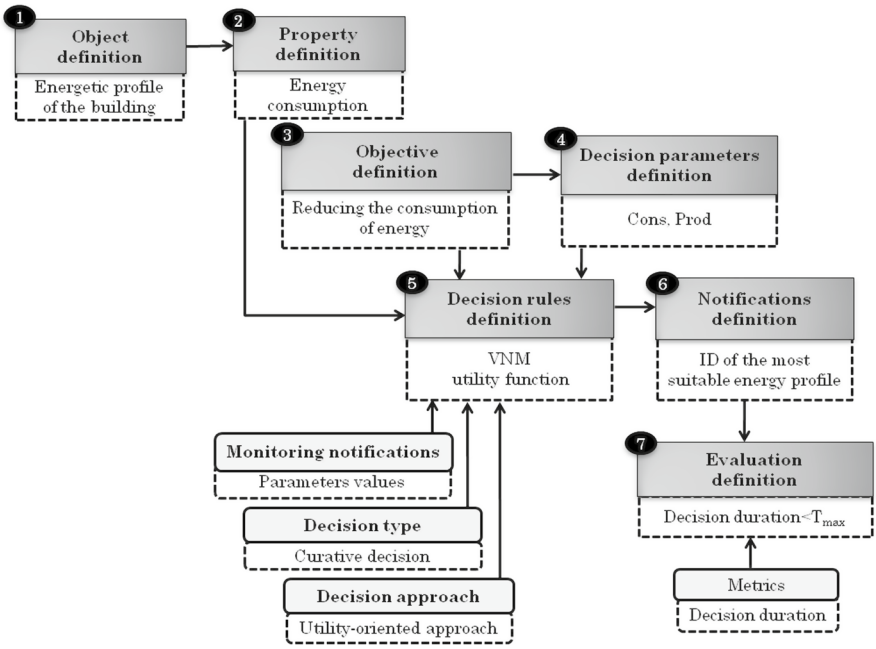

Fig. 5 Instance of the decision guideline for the building energetic profile adaptation

5.1.1 Decision guideline instance for the adaptation of the energetic profile of the building

Figure 5 presents an instance of the design guideline presented in Fig. 2. The decision object is the energetic profile of the building (Step 1). Hence, each alternative has as a type energetic profile, i.e., the set of alternatives correspond to the set of possible energetic profiles of the building. The property of the energetic profile related for this decision is the consumption of energy (Step 2). Consequently, the energetic profiles will be compared according to their consumption of energy. The main objective of the decision consists on reducing the consumption of energy in the building (Step 3). Therefore, the parameters defined for this decision are the energy consumed (Cons) and the energy produced (Prod) by the building (Step 4). The monitoring notifications necessary for this decision are the current values of the parameters Cons and Prod. The type of decision is curative (i.e.., when there is an increase of the energy consumption in the building, the decision will try to decrease this consumption). The chosen decision approach is the Utility-oriented approach. The choice of the Von-Neumann Morgenstern (VNM) utility function as a decision rule (Step 5) is done by following the guidelines for the choice of the utility functions presented in our previous work [2]. The details and the mathematic formula of this utility function are presented in the following paragraph. The notification of decision is the ID of the most suitable energetic profile (Step 6). The decision evaluation

is defined according to the decision duration metric. The decision duration should not exceed T_{max} which is a duration threshold specified by the application designer/administrator (Step 7).

5.1.2 Decision example for the adaptation of the building energetic profile

The SBA ensures the selection of the most suitable energetic profile of the building based on the Von-Neumann Morgenstern utility function, presented by the Eq. 1 (The description of the Von-Neumann Morgenstern utility function is detailed in our previous work [2]).

$$U_{\text{VNM}} = \sum_{k=1}^{3} p_k \times u(S_k) \tag{1}$$

Three situations could appear after making the decision that consists on choosing the most suitable energetic profile: S_1, S_2 and S_3. p_k is the probability that the situation S_k occurs after making the decision, and $u(S_k)$ is the utility that the situation S_k provides to the user.

The first situation S_1 is the situation where the value of the energy consumed by the building's devices (Cons) exceeds the value of the energy produced by the photovoltaic panel of the building (Prod). S_2 is the situation where Cons is equal to Prod. S_3 is the situation where Cons is lower than Prod. The utility values of these three situations must be defined by an expert according to satisfaction degree of the Application User related to each situation. In this example, we consider that the utilities of the situations S_1, S_2 and S_3 are $u(S_1) = -20$, $u(S_2) = 5$ and $u(S_3) = 20$, respectively.

The monitoring notifications (generated from the Monitoring module) indicate that in the current situation of the building the value of the consumed energy exceeds the value of the produced energy, and the current requirements of the user are: switching on the Dishwasher and the Air Conditioners AC_1 (required temperature=21°C) and AC_2 (required temperature=20°C). A decision is needed to choose the most suitable energetic profile. As mentioned in the Sect. 5.1, we consider only two energetic profiles to simplify the presentation of the decision example: the High performance mode energetic profile (A_1) and the Economical mode energetic profile (A_2).

To calculate the utility of each energetic profile, the probability of occurrence of the situations S_1, S_2 and S_3 must be defined by an expert for each energetic profile. For the energetic profile A_1, we consider that the probabilities of S_1, S_2 and S_3 are $p_1 = 0,5$, $p_2 = 0,4$ and $p_3 = 0,1$, respectively. Hence, the utility of A_1 is $U_{\text{VNM}}(A_1) = 0,5 \times (-20) + 0,4 \times 5 + 0,1 \times (20) = --6$. For the energetic profile A_2, the probabilities of S_1, S_2 and S_3 are $p_1 = 0,2$, $p_2 = 0,3$ and $p_3 = 0,5$, respectively. Hence, the utility of A_2 is $U_{\text{VNM}}(A_2) = 0,2 \times (--20) + 0,3 \times 5 + 0,5 \times (20) = 7,5$. Therefore, the energetic profile selected in this example is A_2 since $U_{\text{VNM}}(A_2)$ exceeds $U_{\text{VNM}}(A_1)$.

5.2 Evaluation of our decision approach implementation

In this section, we present some experiments conducted to evaluate our decision approach implementation. Notably, we evaluate the processing time of our decision module, presented in the Sect. 3. All of the experiments were conducted on a laptop equipped with 8GB of RAM and an Intel Core i7 processor rated at 2.2 GHz. The operating system used is Ubuntu, and the Java Virtual Machine version used is 1.6.0.

These experiments consist in calculating the processing time taken by our decision module to make the decision of choosing the most suitable energetic profile of the building based on the Von-Neumann Morgenstern utility function. The experiments consist on the calculation of the processing time while varying the number of alternatives from 20 to 1000 (given as input data to our decision module), for a set of three, five and ten decision parameters, respectively.

For each alternatives number, we repeated the measurement of the decision module processing time many times. By making repeated measurements, we remark that the results of calculation of the processing time values are not the same. This is due to the random error that varies randomly in repeated measurements throughout the conduct of a test [1].

To have a proximate value of the true value of the processing time of each alternatives number, we repeated the experiment Nb times, and we calculated the mean of the processing time measured values M. The resulting population of measurements could be described statistically in terms of the population mean M and the standard deviation S. The random error of these measurements is described by the *random standard uncertainty of the mean*. The higher the number of repetitions of an experiment Nb is, the lower the *random standard uncertainty of the mean* is and the more reliable the value of the mean M (that corresponds to the decision module processing time) is [1]. For an infinite number of experiment repetitions, the *random standard uncertainty of the mean* is equal to zero.

The *random standard uncertainty of the mean* MU is calculated by the Eq. 2.

$$\mathrm{MU} = \frac{S}{\sqrt{Nb}} \tag{2}$$

We repeated the measurement of each experiment Nb times until MU is equal to 0.1, which we consider as an acceptable error value.

Figure 6 presents the evolution of the processing time of our decision module according to the number of possible alternatives. As expected, the experimental results show an increase of the processing time while progressively increasing the number of alternatives. The rate of processing time increase is nearly linear for the three curves (with three, five and ten parameters). Moreover, the results show that the time overhead of the decision module is reasonable, considering the fact that the number of alternatives of most adaptation decisions is unlikely to exceed 1000, and the number of parameters is generally lower than 10.

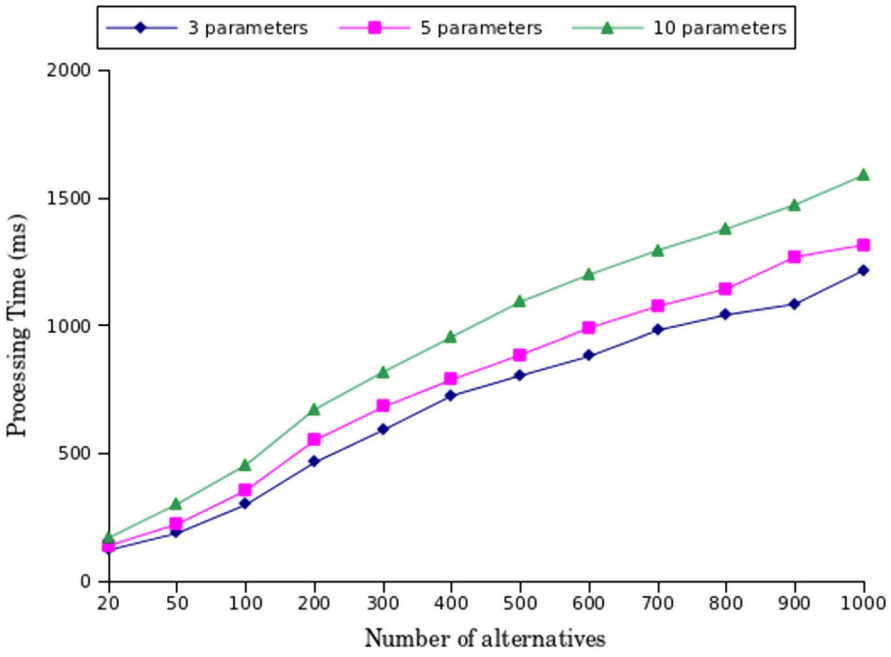

Fig. 6 The evolution of the decision module processing time according to the number of alternatives

6 Discussion and threats to validity

The purpose of this section is to discuss and analyse the different threats to validity regarding our recommendations for the choice of the main existing decision approaches. We also discuss the threats to validity of the implementation of our decision approach.

Our recommendations for the choice of the decision approaches (presented in the Sect. 2.6) are based on the characteristics of each decision approach as well as other characteristics, such as the adaptation frequency, the available resources and the time constraints. The proposed recommendations are able to assist the Application Designer by giving him some generic guidelines. However, they are not able to reduce considerably his effort to choose the most suitable decision approach, since this task needs a deep comprehension of the case study.

The current implementation of our decision approach treats one decision at a considered time, i.e., it does not treat overlapped decision-making processes. We consider that two decision-making processes d_1 and d_2 are overlapped in the case where d_2 needs to be triggered during the execution of d_1. To handle such situations, the decision module can for example choose between:

- Interrupting the execution of d_1 and triggering the execution of d_2 (for example in the case where the decision of d_2 is more critical than the decision of d_1)

- Carrying on the execution of d_1, then triggering the execution of d_2 (for example in the case where the decision of d_1 is more critical than the decision of d_2)

These two management ways of overlapped decision-making processes are not the only ones. In the current version of our decision approach implementation, we suppose that there are no overlapped decision-making processes. However, we expect that our decision module needs an implementation of strategies based on a deep study to handle this kind of situations, especially when our decision module is used by highly dynamic applications.

7 Conclusion

In this work, a novel decision approach for developing context-aware applications in ubiquitous environments was presented. In the literature, we noticed the lack of guidelines that provide a support for applications designers while they are designing the adaptation decisions. Our proposed approach, called DAACS, includes a set of decision recommendations and guidelines to assist the applications designers. DAACS is implemented through a reusable decision module that assists the ubiquitous application developers.

To illustrate the usefulness of our decision approach, we presented a case study, called Smart Building, in which we have applied the guidelines that we proposed in our decision approach. We developed a Smart Building Application for this case study that interacts with our decision module. We showed an illustrative example that demonstrates how to use our decision approach in order to adapt the energetic profile of the building. We also conducted some experiments that show that the processing time overheads introduced by our decision approach implementation to make the adaptation decisions is reasonable, taking into consideration the characteristics (notably the number of alternatives and the number of decision parameters) of most adaptation decisions in a smart building .

As future work, we aim to extend our decision approach implementation by completing the Situation-actions and the Goal-oriented decision approaches implementation, and test adaptation decisions in both Analysis and Execution phases of the MAPE-K loop. We also aim to conduct a thorough study of the decision evaluation and apply more tests of our decision guideline and our decision module, in different case studies, to explore more evaluation metrics.

References

1. American society of mechanical engineers, the performance test standard ptc test uncertainty. Tech. rep. (2005)
2. Abdennadher I, Bouassida Rodriguez I, Jmaiel M. A Utility-Based Approach for Self-Adaptive Systems: Application to a Smart Building. In: 14th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2017, Hammamet, Tunisia, October 30 - Nov. 3, 2017, pp. 76–82

3. Abdennadher I, Khabou N, Bouassida Rodriguez I, Jmaiel M. Designing Energy Efficient Smart Buildings in Ubiquitous Environments. In: 2015 15th International Conference on Intelligent Systems Design and Applications (ISDA), pp. 122–127 (2015)
4. Aldini A (2018) Design and verification of trusted collective adaptive systems. Trans Model Computer Simul 28(2):1–27
5. Ben Alaya M, Matoussi S, Monteil T, Drira K. Autonomic computing system for self-management of machine-to-machine networks. In: Proceedings of the 2012 International Workshop on Self-aware Internet of Things, pp. 25–30 (2012)
6. Elmalaki S, Wanner L, Srivastava M. Caredroid: Adaptation Framework for Android Context-Aware Applications. In: Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, pp. 386–399 (2015)
7. Francoise A, Erwan D, Guillaume G. Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures. In: Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on, pp. 309–314 (2010)
8. Gauvrit G, Daubert E, André F. Safdis: A Framework to Bring Self-Adaptability to Service-Based Distributed Applications. In: Proceedings of the 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA '10, pp. 211–218 (2010)
9. Kakousis K, Paspallis N, Papadopoulos GA (2010) A survey of software adaptation in mobile and ubiquitous computing. Enterp Inf Syst 4(4):355–389
10. Kallel S, Charfi A, Mezini M, Jmaiel M. Combining Formal Methods and Aspects for Specifying and Enforcing Architectural Invariants. In: Proceedings of the 9th International Conference on Coordination Models and Languages COORDINATION, *Lecture Notes in Computer Science*, vol. 4467, pp. 211–230 (2007)
11. Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 36(1):41–50
12. Klein C, Schmid R, Leuxner C, Sitou W, Spanfelner B. A Survey of Context Adaptation in Autonomic Computing. In: Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08), pp. 106–111 (2008)
13. Klös V, Göthel T, Glesner S. Adaptive Knowledge Bases in Self-Adaptive System Design. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, pp. 472–478 (2015)
14. Krupitzer C, Roth FM, VanSyckel S, Schiele G, Becker C (2015) A survey on engineering approaches for self-adaptive systems. Pervasive Mobile Comput 17:184–206
15. Kuze N, Kominami D, Kashima K, Hashimoto T, Murata M (2018) Self-organizing control mechanism based on collective decision-making for information uncertainty. ACM Trans Auton Adapt Syst 13(1):1–21
16. Lei Y, Ben K, He Z. A Model Driven Agent-Oriented Self-Adaptive Software Development Method. In: 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 2242–2246 (2015)
17. Lesk M, Kernighan B. Computer Typesetting of Technical Journals on UNIX. In: Proceedings of American Federation of Information Processing Societies: 1977 National Computer Conference, pp. 879–888. Dallas, Texas (1977)
18. Mira Vrbaski Dorina Petriu D.A. Tool Support for Combined Rule-Based and Goal-Based Reasoning in Context-Aware Systems. In: 2012 IEEE 20th International Requirements Engineering Conference (RE), pp. 335–336 (2012)
19. Moreno GA, Cámara J, Garlan D, Schmerl B. Proactive Self-Adaptation Under Uncertainty: A Probabilistic Model Checking Approach. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 1–12 (2015)
20. Morvaj B, Lugaric L, Krajcar S. Demonstrating Smart Buildings and Smart Grid Features in a Smart Energy City. In: Proceedings of the 2011 3rd International Youth Conference on Energetics (IYCE), pp. 1–8 (2011)
21. Pascual GG, Pinto M, Fuentes L (2015) Self-adaptation of mobile systems driven by the common variability language. Fut Gener Computer Syst 47:127–144 (**Special Section: Advanced Architectures for the Future Generation of Software-Intensive Systems**)
22. Paucar L, Bencomo N. A Survey on Preferences of Quality Attributes in the Decision-Making for Self-Adaptive Systems: The Bad, the Good and the Ugly. In: CIbSE 2017 - XX Ibero-American Conference on Software Engineering, pp. 1–14 (2017)

23. Peng X, Chen B, Yu Y, Zhao W. Self-Tuning of Software Systems Through Goal-Based Feedback Loop Control. In: 2010 18th IEEE International Requirements Engineering Conference, pp. 104–107 (2010)
24. Sancho G. Adaptation d'architectures Logicielles Collaboratives Dans Les Environnements Ubiquitaires. Contribution à l'interopérabilité par la sémantique. Ph.D. thesis, Université Toulouse 1 Capitole (UT1 Capitole) (2010)
25. Schilit BN, Theimer MM (1994) Disseminating active map information to mobile hosts. IEEE Netw 8(5):22–32
26. Wei EJ, Chan AT (2013) Campus: a middleware for automated context-aware adaptation decision making at run time. Pervasive Mobile Comput 9(1):35–56
27. Weiser M (1991) The computer for the 21st century. Scientif Am 265:66–75
28. Zarghami A, Sapkota B, Eslami MZ, van Sinderen M. Decision As a Service: Separating Decision-Making from Application Process Logic. In: 2012 IEEE 16th International Enterprise Distributed Object Computing Conference, pp. 103–112 (2012)
29. Zhao T. The Generation and Evolution of Adaptation Rules in Requirements Driven Self-Adaptive Systems. In: 2016 IEEE 24th International Requirements Engineering Conference (RE), pp. 456–461 (2016)
30. Zimmermann O. Architectural Decision Identification in Architectural Patterns. In: Proceedings of the WICSA/ECSA 2012 Companion Volume, pp. 96–103 (2012)