Cost-Benefit Analysis at Runtime for Self-adaptive Systems Applied to an Internet of Things Application

M. Jeroen Van Der Donckt[1], Danny Weyns[1,2], M. Usman Iftikhar[1,2] and Ritesh Kumar Singh[1]
[1]*Department of Computer Science, KU Leuven, Belgium*
[2]*Department of Computer Science, Linnaeus University, Växjö, Sweden*

Keywords: Self-adaptation, MAPE, Models at Runtime, Statistical Model Checking, Cost-Benefit Analysis Method, CBAM, Internet-of-Things, IoT.

Abstract: Ensuring the qualities of modern software systems, such as the Internet of Things, is challenging due to various uncertainties, such as dynamics in availability of resources or changes in the environment. Self-adaptation is an established approach to deal with such uncertainties. Self-adaptation equips a software system with a feedback loop that tracks changes and adapts the system accordingly to ensure its quality goals. Current research in this area has primarily focussed on the benefits that self-adaptation can offer. However, realising adaption can also incur costs. Ignoring these costs may invalidate the expected benefits. We start with demonstrating that the costs for adaptation can be significant. To that end, we apply a state-of-the-art approach for self-adaptation to an Internet of Things (IoT) application. We then present CB@R (Cost-Benefit analysis @ Runtime), a novel model-based approach for runtime decision-making in self-adaptive systems. CB@R is inspired by the Cost-Benefit Analysis Method (CBAM), which is an established approach for analysing costs and benefits of architectural decisions. We evaluate CB@R for a real world deployed IoT application and compare it with the conservative approach applied in practice and a state-of-the-art self-adaptation approach.

1 INTRODUCTION

Modern software systems often operate in very dynamic environments. Examples are IoT systems, such as unmanned underwater vehicles (UUVs) that are used for oceanic surveillance to monitor pollution levels (Gerasimou et al., 2017), and supply chain systems that ensure sufficient, safe, and nutritious food to the global population (Bennaceur et al., 2016). The dynamics of such systems introduce uncertainties that may be difficult or even impossible to anticipate before deployment. Hence, these systems need to resolve the uncertainties during operation.

A common technique to deal with uncertainties in modern software systems is self-adaptation (Salehie, 2009; Cheng et al., 2009; De Lemos et al., 2013; Weyns, 2018; de Lemos et al., 2017) that equips a system with one or more feedback loops. The aim of self-adaptation is to let the system collect additional knowledge about itself and its environment, and adapt itself to satisfy its goals under the changing conditions, or if necessary degrade gracefully.

Current self-adaptation approaches tend only to look at the benefits that can be gained by adaptation (Weyns and Ahmad, 2013). These benefits are typically improvements in terms of system qualities (self-optimisation, self-healing, etc. (Kephart and Chess, 2003)). However, applying adaptations to a system may also incur costs. These costs are domain specific and may be expressed in terms of the extra use of resources or the time or energy that it takes to apply the adaptation actions. For example, in an IoT system with battery-powered motes, the cost may be expressed as the energy required to communicate adaptation actions to the motes that need to adapt their network settings.

Examples of initial work in this direction are (Bertolli et al., 2010) that uses an adaptation cost model for quantifying the overhead introduced by autonomic behaviour in an adaptive parallel application, and (Cailliau and van Lamsweerde, 2017) that performs a tradeoff analysis in a runtime adaptation approach to increase the satisfaction rate of system goals. However, to the best of our knowledge, no systematic approach exists today that considers both the benefits and costs for adaptation as first-class citizens in the decision-making for self-adaptation. Hence, the research problem tackled in this paper is:

Van Der Donckt, M., Weyns, D., Iftikhar, M. and Kumar Singh, R.
Cost-Benefit Analysis at Runtime for Self-adaptive Systems Applied to an Internet of Things Application.
DOI: 10.5220/0006815404780490
In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, pages 478-490
ISBN: 978-989-758-300-1

How to enable decision-making in self-adaptive systems that considers both the benefits of adaptation and the costs for realising adaptation as first-class concerns?

To underpin the relevance of the research question, we start with demonstrating that the costs for adaptation can be significant. To that end, we apply a state-of-the-art approach for self-adaptation (Iftikhar and Weyns, 2014) to DeltaIoT, an Internet of Things (IoT) application that has been proposed as an exemplar for research in the field of engineering self-adaptive systems (Iftikhar et al., 2017).

To tackle the research problem of this paper, we then introduce CB@R (Cost-Benefit analysis @ Runtime), a new cost-benefit analysis method for decision-making in self-adaptive systems that exploits models at runtime (Blair et al., 2009). CB@R leverages on the principles of the Cost-Benefit Analysis Method (CBAM), which is an established approach for analysing costs and benefits of decisions in architectural design (CBAM, 2018). CB@R considers both the expected benefits produced by adaptation and the expected cost implied to realise adaptation as first class concerns when selecting a configuration to adapt the system and realise the adaptation goals.

We evaluate CB@R with a real world setup of an IoT application, called DeltaIoT (Iftikhar et al., 2017), and compare it both with a conservative approach that is applied in practice and a state-of-the-art adaptation approach.

The remainder of this paper is structured as follows. Section 2 provides background on CBAM, the basics of self-adaptation, and DeltaIoT. In Section 3, we illustrate the relevance of costs for adaptation with a state-of-the-art self-adaptation approach applied to DeltaIoT. Section 4 introduces CB@R, the novel cost benefit analysis approach for decision-making at runtime in self-adaptive systems. In Section 5, we evaluate CB@R and compare it with two other approaches. Section 6 discusses related work. Finally, Section 7 draws conclusions and outlines ideas for future work.

2 BACKGROUND

2.1 CBAM

The Cost Benefit Analysis Method (CBAM) is an established method for analysing the costs and benefits of architectural designs of software-intensive systems (CBAM, 2018). CBAM takes into account the uncertainty factors regarding costs and benefits, providing a basis for informed decision-making about architectural design or upgrade. The concepts of CBAM are used as inspiration for CB@R.

Given that the resources for building and maintaining a software-intensive system are finite, software architects require a rational process to help them choose among architectural options. An architectural option refers to a candidate design of a software-intensive system that is based on applying a particular architectural approach (for example a peer-to-peer style or a client-server style). Different *architectural options* will have different technical and economic implications. Technical implications are the various implemented features and the qualities associated with them, each of which brings some benefit to the organisation. A direct economic implication is the cost of implementing the system.

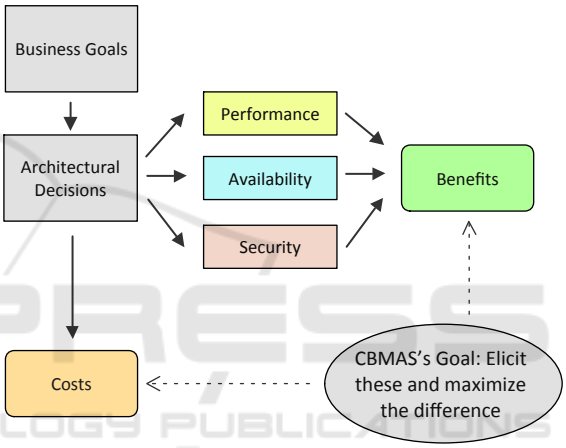

Figure 1: Context for CBAM process.

CBAM models the benefits and costs of architectural design decisions providing a mean for optimising such decisions. We illustrate CBAM functionality by using Figure 1. In the example, the relevant qualities may be performance, availability and security, each with a particular importance. Each architectural design (resulting from a set of architectural decisions) will yield different values for these qualities, resulting in some benefit for the system stakeholders. Each architectural design will also incur a cost. For example, using redundant hardware to achieve a desired level of availability has a cost, while check pointing to a disk file has a different cost. As shown in the figure, the goal of CBAM is to maximise the difference between the benefit derived by the system design and the cost of implementing the design.

Using the benefit and cost of each architectural design, CBAM allows the stakeholders choosing among architectural designs based on their Value For Cost (VFC), the ratio of utility to cost:

$$VFC = \frac{TotalBenefit}{Cost} \quad (1)$$

TotalBenefit stands for the overall utility produced by an architectural design. This total benefit is defined as the weighted sum of the utility values for the examinated quality attributes. The VFC values are used to rank the architectural designs (and thus design decision), from which the stakeholders can choose.

2.2 Self-adaptation

In this research, we apply architecture-based self-adaptation (see Figure 2), where a self-adaptive system consists of a Managed System that operates in an Environment providing the domain functionality to users of the system and a Managing System that monitors and adapt the Managed System realising some Adaptation Goals (Oreizy et al., 1998; Garlan et al., 2004; Kramer and Magee, 2007; Weyns et al., 2012).

At a given time, the Managed System has a particular configuration that is determined by the arrangement and settings of its running components. We refer to the different choices for adaptation from a given configuration as the *adaptation options*. Adapting the managed system means selecting an adaptation option and changing the current configuration accordingly. The Environment in which the system operates can be the physical world or computing elements that are not under control of the system. The Environment and the Managed System may expose stochastic behaviour.

A common approach to realise the Managing System is by means of a MAPE-K feedback loop (Kephart and Chess, 2003; Dobson et al., 2006; Salehie, 2009; Weyns et al., 2013) that comprises four components: Monitor, Analyse, Plan, and Execute that share common Knowledge. Knowledge comprises models of the managed system referring to its structure, behaviour, goals, and other relevant aspects of the system or its adaptation (Weyns et al., 2012). The Monitor collects runtime data from the Environment and the Managed system to keep the models up to date. The Analyser analyses the qualities of the different adaptation options and compares these with the qualities of the current configuration to determine whether adaptation is required. If so, the Planner select the best adaptation option based on the adaptation goals and composes a plan to adapt the system. Finally, the Executer executes the plan.

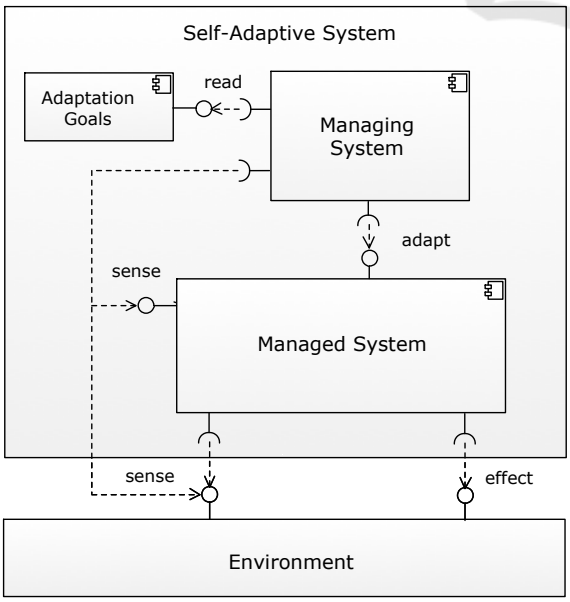

Figure 2: Basic conceptual model of self-adaptive system (Weyns, 2018).

2.3 DeltaIoT

DeltaIoT is a reference Internet of Things application (IoT) that has been developed to evaluate new approaches to self-adaptation and compare their effectiveness with other solutions (Iftikhar et al., 2017). DeltaIoT compromises both a simulator for offline experimentation as well as real physical setup.

DeltaIoT has been set up by VersaSense, a provider of industrial IoT products and services (www.versasense.com/). The network consists of 15 Long-Range IoT motes (LoRa) that are deployed at the campus of KU Leuven as shown in Figure 3. The motes are strategically placed to provide access control to labs (via RFID sensor), to monitor the movements and occupancy status (via passive infrared sensor) and to sense the temperature (via heat sensor). Each mote in the network relays its sensor data to the gateway using wireless multi-hop communication. Communication in the network is time-synchronised and organised in cycles. Each cycle comprises a fixed number of communication slots. Each slot defines a sender mote and a receiver mote that can communicate with one another. The communication slots are fairly divided among the motes. For example, the system can be configured with a cycle time of 570 seconds (9.5 minutes), each cycle comprising 285 slots, each of 2 seconds. For each link, 40 slots are allocated for communication between the motes.

Two key qualities of DeltaIoT are energy consumption and reliability. Since motes are battery powered and sending messages is the dominant energy cost, it is crucial for the system to optimise communication. Two factors determine the critical quality properties: the transmission power settings used by the motes for communication (ranging from 1 for min power to 15 for max power) and the distribution of

Figure 3: DeltaIoT deployment at KU Leuven Campus.

the messages sent by each mote over the links to its parents (e.g., for motes with two parents: 0% to one parent and 100% to the other, 20/80, 40/60, 60/40, 80/20 and 100/0). DeltaIoT offers a user interface for operators to set both the power settings of the motes and the distribution of messages in the network. This interface shows the status of user defined properties.

The DeltaIoT network is subject to various uncertainties. We consider in this paper two types of uncertainties: network interference (e.g. due to activities of neighbouring computing systems, or due to changing whether conditions) and fluctuating load of messages (the load of messages generated by motes may depend on various factors, such as the presence of people). These uncertainties make guaranteeing the system required properties a complex problem. To deal with these uncertainties, current practice typically applies a conservative approach, where the transmission power is set to maximum and all messages are forwarded to all parents. Operators may tune some of the settings based on experiences. This approach guarantees high reliability, but implies high energy consumption and thus reduced lifetime of the network.

For the evaluation with DeltaIoT, we define two concrete quality requirements that need to be realised regardless of possible network interference and fluctuating load of messages generated in the network:

R1: The average packet loss over 15 hours should not exceed 10%.

R2: The average energy consumption over 15 hours should be minimized.

When self-adaptation is applied, these two quality requirements become the adaptation goals.

3 RELEVANCE OF COSTS FOR ADAPTATION

First we demonstrate the relevance of the cost for adaptation using an example. Concretely, we apply ActivFORMS (Active FORmal Models for Self-adaptation) (Iftikhar and Weyns, 2014), a formally founded-model driven approach for self-adaptation, to the DeltaIoT exemplar. ActivFORMS applies architecture-based self-adaptation, where the managing system is realised with a MAPE-K feedback loop. The Knowledge compromises a runtime model for each quality property that is subject of adaptation (corresponding to the adaptation goals). Each quality model is specified as a stochastic timed automaton (or network of these). The monitor updates variables in these models that represent uncertainties using data collected at runtime. The Analyzer dynamically computes the adaptation options based on the variability in transmission power settings of the motes and the distribution factors for the links in the network (see Section 2.3). For each adaptation option, the Analyser estimates the expected qualities by running a number

of simulations on the quality models. The results for the qualities of the adaptation options are used as selection criteria in the Planner. In particular, the Planner selects the best option by applying rules that express the system quality requirements for DeltaIoT. For the best option a plan is composed. This plan defines the adaptation actions per mote (change power setting and/or distributed factor) that are required to adapt the current configuration to the configuration of the selected best adaptation option. The Executor then executes the actions of the plan.

When applied to DeltaIoT, the approach described above only considers the benefits of adaptation (i.e. the contribution of adaptation to the adaptation goals). To determine the importance of the cost for realising adaptations, we start with defining this cost. In DeltaIoT, there is an upstream communication of messages with data from the motes to the gateway. The benefits are defined for this upstream communication. The adaptation actions are sent downstream, from the gateway to the motes. We define the cost as the energy that is consumed for sending and receiving the adaptation message downstream. Default, ActivFORMS does not take into account this cost when selecting adaptation options. We study the impact of the cost based on two definitions:

$$CostV1 = \sum_{x} pLenA(x) * R + (pLenA(x) - 1) * S \quad (2)$$

$$CostV2 = \sum_{y} pLenL(y) * R + (pLenL(y) - 1) * S \quad (3)$$

With x referring to a distinct adaptation action (e.g. change the power setting of mote m_7 along the link to m_2 to 6), while y refers to all the adaptation actions of a link (e.g. change the power of mote m_7 along the link to mote m_2 to a setting of 6, and change the distribution factor of this link to 40%). R and S refer to the energy consumption of receiving or sending a message respectively. *pLenA()* and *pLenL()* return the length of the path from the gateway to the source mote of the link that is subject of adaptation. Since the gateway is directly connected to the electricity net, the cost of downstream communication of the first link is not relevant; this explains the minus symbol in both definitions. In the first version of the cost, the information for each adaptation is (naively) sent as a separate message; in the second definition, adaptations per link are sent in one message.

To determine the impact of the cost for adaptation, we compute the total energy cost as follows:

$$TotEnergyConsV_i = EnergyCons + CostV_i \quad (4)$$

Where i can be either 1 or 2 referring to one of the cost functions. *EnergyCons* refers to the energy consumption for downstream messages, while $CostV_i$ refers to the energy required for transmitting the adaptation actions upstream. Hence, the total energy consumption is defined as the sum of both.

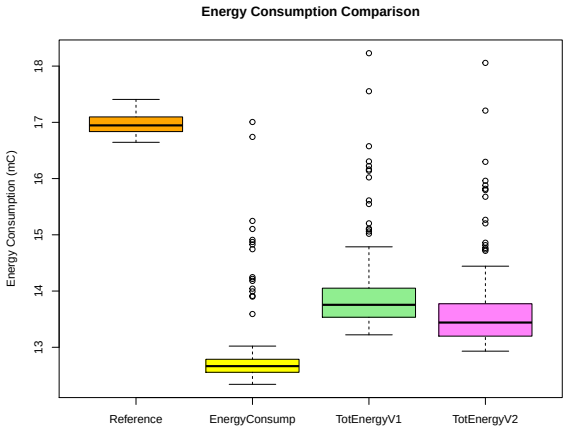

Figure 4: Impact of cost for ActivFORMS on DeltaIoT.

To determine the impact of the cost for adaptation, we test the following hypothesis:

H_T : *The total energy cost for communication, including up- and downstream, is higher as the cost to transmit the message with data to the gateway.*

We used a setup with 15 motes (as shown in Figure 4) and 256 adaptation options. The IoT network was initialised using the reference configuration (max. power settings and duplication of messages sent to all parent motes). We performed four runs of 96 cycles on the simulator. Figure 4 shows the results.

When comparing the energy consumption using plain ActivFORMS (*EnergyCons*) with the approaches that take into account the cost for adaptation ($TotEnergyConsV_i$), we observed a statistically significance difference (p-values smaller as $2.2 * 10^{-16}$ using Wilcox rangsum test). For the first definition of cost the average reduction of energy consumption is 5.60% (mean 14.07 compared to 12.98 for plain ActivFORMS), while for the second definition the average reduction of energy consumption is 7.75% (mean 13.75). In practice, the increase of the lifetime of the network may be smaller as not all motes consume the same amount of energy over time, but of course this applies to all evaluated approaches. Since the total cost for V_2 is lower as for V_1, we use the V_2 definition for the evaluation in Section 5.

To conclude, we can accept hypothesis H_T. For ActivFORMS, a state-of-the-art adaptation approach, applied to the DeltaIoT exemplar, realising adaptation has a significant cost and cannot be ignored.

4 CB@R

We now present a novel method, called CB@R (Cost-Benefit analysis @ Runtime). CB@R enables runtime decision-making in self-adaptive systems that explicitly considers benefits and costs. The approach is inspired by CBAM. However, in contrast to CBAM, CB@R is an automated approach that makes (architectural) reconfiguration decisions of the system at runtime to deal with uncertainties. Hence, CB@R can be considered as a runtime extension of CBAM, where reconfiguration of the system is necessary during operation to guarantee the best Value For Cost (VFC). Figure 5 shows how CB@R is integrated in a MAPE feedback loop.

Knowledge Runtime Models. Central to the realisation of CB@R are a set of runtime models maintained in the Knowledge repository of the MAPE feedback loop. The models include a *system model* that provides an abstraction of the managed system relevant to realise the adaptation goals, and a *context model* that provides an abstraction of the relevant parts of the environment in which the system operates.

Next, the Knowledge repository contains a *quality model* for each quality property that is subject to adaptation. These quality models (possibly combined with a system model and/or a context model) allow a runtime verifier to estimate the quality properties for a particular configuration of the managed system (i.e., an adaptation option as explained next).

To make an adaptation decision, CB@R requires a set of *adaptation options*. Each adaptation option defines a particular configuration of the system that can be reached through adaptation from the current configuration. The set of adaptation options defines all such possible configurations of the managed system. In this research, we assume that the managed system has a limited, potentially high number of adaptation options. This set may dynamically change over time. This implies that system parameters with a continuous domain that determine configurations need to be discretised. Heuristics can be applied to select the adaptation options in case of a very large set, but this is out of scope of this paper. Note that this assumption is not a restriction of CB@R in itself.

To determine the benefits of adaptation with CB@R, a set of runtime *utility response curve models* are required, one for each quality property that is subject of adaptation. A utility response curve model (Len et al., 2013) for a particular quality expresses the utility for the range of possible values of that quality property. In a response curve model, the utility values can range from 0 to 100, while the values for the qualities range from relevant minimum to maximum values of the quality property. The utility response curve models are defined in consultation with the stakeholders and express how the utility for different values of the quality property changes as valued by the stakeholders. Hence, capturing the utilities of alternative responses of adaption options enables CB@R to make trade-off decisions in relation to the different quality properties. Utility response curves can vary linearly, nonlinearly, as a step function, or combinations of these.

To determine the cost of adaptation with CB@R, the approach requires a *cost model*. Cost models are domain specific and may represent various implications of realising adaptation, such as delay caused through adaptation, resources required to realise adaptation, etc. The cost model should allow to determine the cost for each adaptation option, so that CB@R can perform a cost-benefit analysis for the different adaptation options.

Finally, the Knowledge repository contains a *plan* that is produced by the Planner after an adaptation decision is made. A plan consists of the adaptation actions that are required to adapt the system as required.

Realisation of CB@R. The MAPE elements exploit the runtime models to realise CB@R as follows. The *Monitor* tracks quality properties and uncertainty parameters of the managed system and the environment. The collected data is used to update the system model, the context model, and the different quality models (1). When the *Analyzer* is triggered (2) it determines the adaptation options (3), i.e., all relevant configurations of the managed system that can be reached through adaptation from the current configuration. For each adaptation option, the Analyzer determines the expected qualities that are relevant for adaptation using the *Runtime Verifier* (4.1). To that end, the verifier uses the runtime models of the different qualities, possibly combined with the system and context models, and computes estimates for the different qualities of each adaptation option (4.2). The verification results, i.e., the expected values of the quality properties associated with the different adaption options, are then updated in the Knowledge repository (5).

Next (6), the *Planner* determines the Value For Cost (VFC) for each adaptation option (7). To calculate the total benefit of an adaptation option the Planner first computes the utility for each quality. To that end, the Planner uses the estimates for each quality as determined by the Analyzer. The Planner determines the utility for each estimated quality property using the utility response curves model for the quality. This is repeated for each quality that is subject of adaptation. As different quality attributes will have different

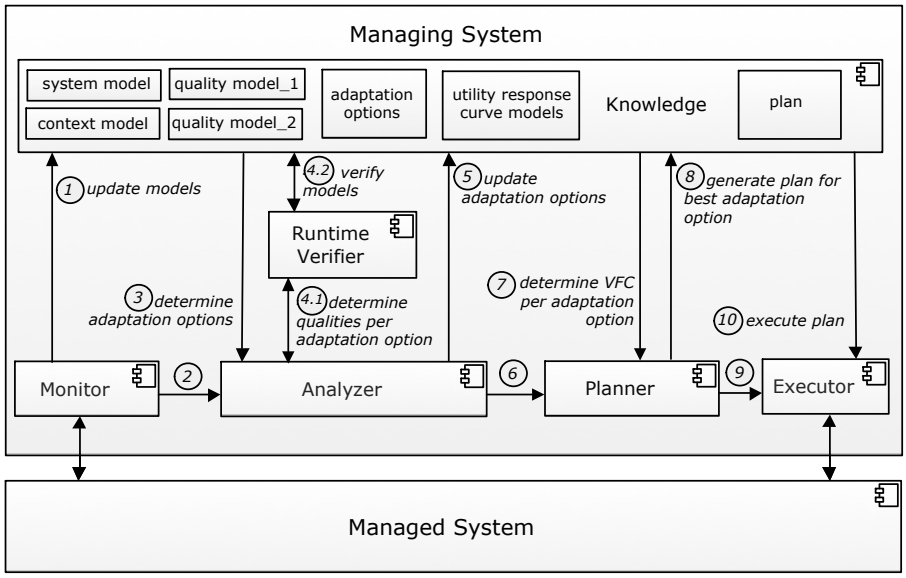

Figure 5: MAPE feedback loop with runtime decision-making using CB@R.

importance to the stakeholders, each quality is assigned a weight (the weights express the relative importance of a qualities, so the sum of the weights should be equal to one). The Planner determines the expected overall benefit B_i for each adaptation option i by summing the utility associated with each quality weighted by the importance of that quality attribute as follows:

$$B_i = \sum_j (U_j(x_{(i,j)}) - U_j(c_j)) * W_j \quad (5)$$

Where the Planner takes the sum over each quality j, with $x_{i,j}$ the value for the j^{th} quality of the i^{th} adaptation option and $U_j(x_{(i,j)})$ the expected utility for this quality (determined by the utility response curve model). C_j is the value for the j^{th} quality in the current configuration and $U_j(c_j)$ the corresponding utility. W_j is the weight for the quality. To determine the expected benefit for an adaptation option, the Planner takes the sum of the difference between the expected utility and the current utility for each quality taking into account the respective weights.

Next, the Planner uses the domain-specific cost model to determine the expected cost for each adaptation option. With the estimates for the benefits and the costs for each adaptation option, the Planner can finally calculate the VFC. As introduced in Section 2, for each adaptation option, VFC is defined as the ratio benefit B_i, to cost C_i:

$$VFC_i = \frac{B_i}{C_i} \quad (6)$$

The values for VFC are then used to rank the adaptation options under consideration. CB@R selects the adaptation option with the highest VFC value. For this option, the Planner generates a plan (8) and updates the Knowledge repository. Finally, when the *Executor* is triggered (9), it starts executing the plan produced by the Planner, adapting the managed system according to the selected adaptation option (10).

Example – We illustrate the principles of CB@R by applying them to the DeltaIoT exemplar.

The two adaptation goals in DeltaIoT correspond with the two requirements: energy consumption and packet loss. Figure 6 shows the runtime model that the Analyzer uses to estimate energy consumption.

The verifier uses this model to estimate the energy consumption for the different adaptation options. In DeltaIoT the adaptation options are determined by the power settings used by the motes to communicate messages and the distribution of messages sent to parent motes (see Section 2.3). Hence, the parameters of the models are configured for each adaptation option accordingly. Furthermore, the values of the parameters that represent uncertainties tracked by the Monitor are set; e.g., the values of *pTraffic(moteID)* represents the traffic that a mote with a given *id* is expected to generate. Once the settings for an adaptation option are set, the estimated energy consumption is determined. To that end, the System automaton activates the motes one by one *moteId = nextTurn()*. Each Mote can then send messages to its parents in the time slots dedicated to it (*sendPackets(packets)*). The energy required to send the messages is then computed (*calcSendEnergy(packets)*). When the Gateway gets its turn, it computes the total energy consumed by the motes both to send and receive messages. This

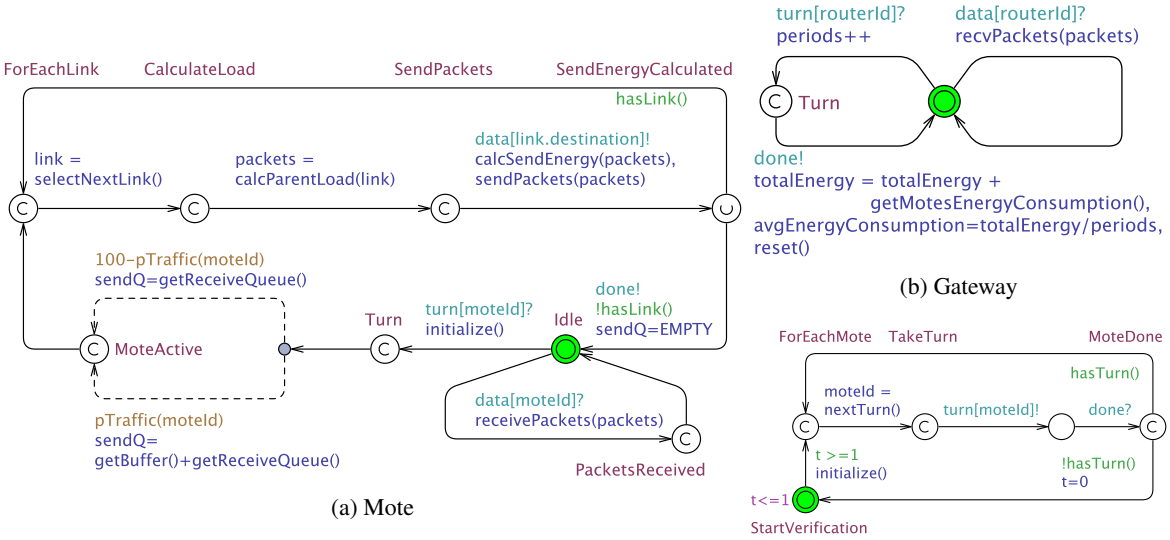

Figure 6: Runtime model to estimate energy consumption for different adaptation options.

process is repeated for 30 times to compute and estimated average energy consumption with a required accuracy. For each simulation run the verifier assigns uncertainty values for pTraffic(moteId) per mote.

The adaptation goals are defined with two utility-response curve models, one for each quality of interest as shown in Figure 7 and Figure 8.

The utility response curve model for energy consumption was defined in consultation with engineers of VersaSense based on the results of the self-adaptive solution presented in the previous section. From the simulation runs on DeltaIoT we know that the mean value of the energy consumption is around 13mC. This explains the first vertical drop on the graph. Values between 13mC and 12mC are progressively considered better, while any value below 12mC is consider excellent. Values between 13mC and 15mC are acceptable, but poor. Any value above 15mC is considered inferior. The utility response curve model for the packet loss was defined in a similar way with the VersaSense staff. The utility slowly decreases between 0% and 5% packet loss. From 5% onwards, the utility decreases faster and values above 10% are considered inferior.

The cost model in DeltaIoT is determined by the energy that is required to send the adaptation messages from the gateway to the respective motes. We explained this in detail in Section 3.

To illustrate the computation VFC, consider that we have two adaptation options, one with 2% as estimated value for the packet loss and one with 9%, both satisfying the first quality requirement of DeltaIoT. Let us assume that the estimated energy consumption for both options are about the same. In CB@R, the expected utility induced by the 2% packet loss (around 90% see Figure 8) will have a substantial influence on the total benefit of the adaptation option being considered (for packet loss 9% the expected uti-

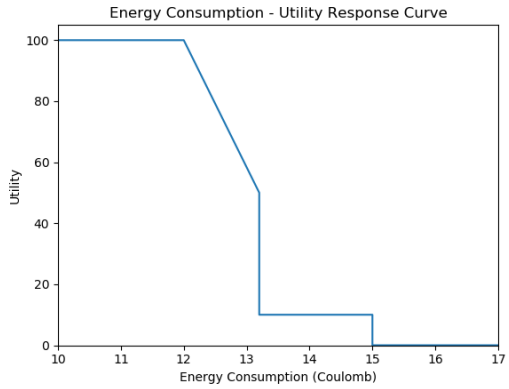

Figure 7: Utility response curve for energy consumption.

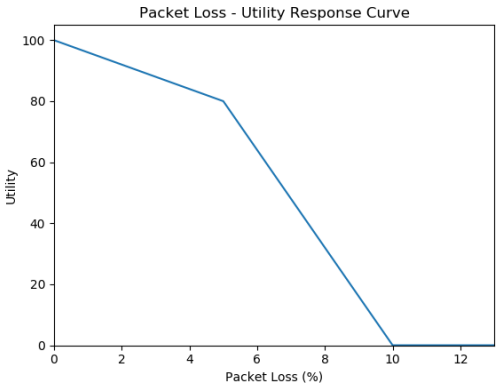

Figure 8: Utility response curve model for packet loss.

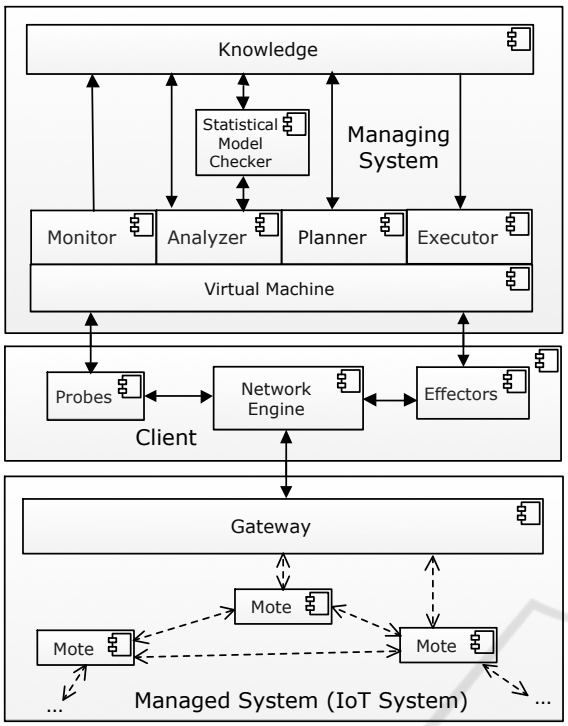

Figure 9: CB@R applied to DeltaIoT.

lity will be around 10%). In case the adaptation costs for both options are similar, this will lead to a higher VFC value for the option with 2% packet loss, and consequently this configuration will be higher ranked than the 9% option. However, in case the cost for adaptation of the 2% option would be substantially higher, this option may not be selected. Compared to rule-based approaches this offers significant flexibility for decision-making.

Prototype realisation – Figure 9 shows the architecture of the self-adaptive DeltaIoT system realisation.

The *Managed System* that consists of the network of motes connected to the gateway is connected to a *Client* that provides an interface to the system. The client offers an interface to the IoT system to monitor various parameters and perform adaptations.

The *Managing System* is deployed on top of the client. For the realisation of CB@R we have used networks of timed automata (TA) for the specification of MAPE feedback loop. We used a set of model templates for the specification and the verification of the correctness of the MAPE feedback loop models (Iglesia and Weyns, 2015). We applied the approach presented in (Iftikhar and Weyns, 2014) that allows deploying and directly executing these models to realise self-adaptation at runtime using a virtual machine, as shown in Figure 9. The Analyzer in the DeltaIoT realisation uses a *Statistical Model Checker* as verifier to make estimates for the different qualities of the adaption options. Figure 10 shows the planner model used in the realisation. The planner waits in the Waiting state to be triggered by the analyser (*plan?*). It then selects the best adaptation option (*selectBestAdaptationOption()*). The pseudo code of this function, which is at the heart of the CB@R method, is shown in Figure 1. If a valid adaptation option is found a plan is created (per link of the DeltaIoT network a set of *ChangePower* and *ChangeDistribution* actions are composed). If no valid option is found a failsafe strategy is applied. When the plan is ready, the executer model is triggered to execute the plan (*execute!*). We provide all the models, verification properties and results, the implementation and the evaluation here: https://people.cs.kuleuven.be/danny.weyns/software/ActivFORMS/.

5 EVALUATION

For the evaluation of CB@R we used the physical setup of the DeltaIoT network deployed at the KU Leuven Campus, which is remotely accessible for use (Iftikhar et al., 2017). We used a standard setting with 15 motes as shown in Figure 3 with communication cycles of 9.5 minutes. We compared three approaches: a conservative approach applied by VersaSense in similar application domains (referred to as reference approach), a rule based approach (that takes into account cost for adaptation similar to what we discussed in Section 3), and CB@R (we applied equal weights for the quality properties). Each approach ran for 15 hours on the physical network, i.e., 90 communication cycles. In each cycle, the network can be adapted. As mentioned in the previous section, the two quality requirements are used as adaptation goals: (1) keep the average packet loss under 10% and (2) minimise the average energy consumption, both over 15 hours. We test the following hypothesis:

H_1 : *The adaptation approaches outperform the reference approach for energy consumption*

H_{21} : *The rule-based approach achieves the packet loss requirement*

H_{22} : *CB@R achieves the packet loss requirement*

H_{31} : *CB@R has significantly better results than a rule-based approach for the average energy consumption*

H_{32} : *CB@R has significantly better results than a rule-based approach for average packet loss.*

First, we look at energy consumption. Figure 11 shows the results of the three approaches. The mean

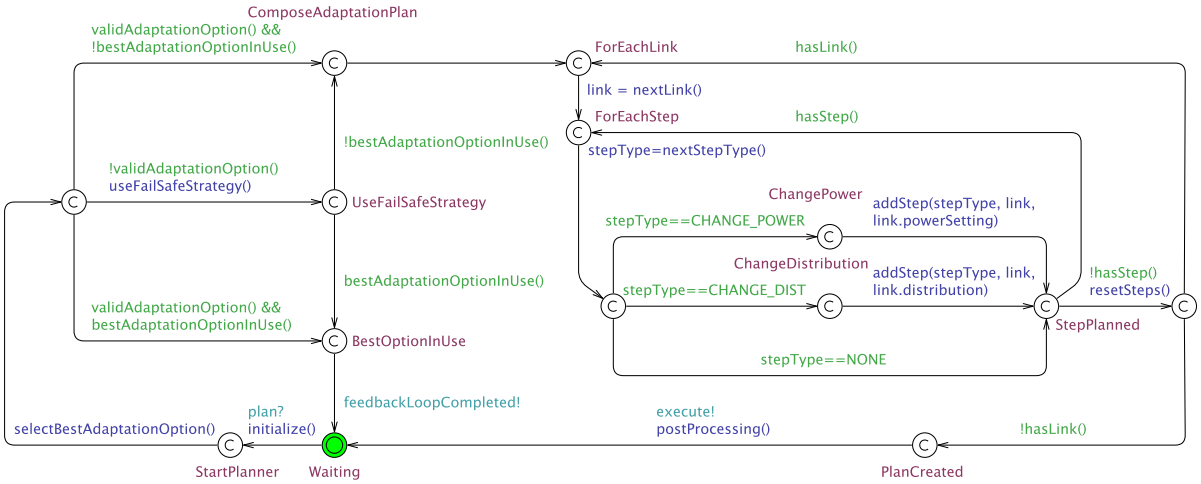

Figure 10: Planner model for CB@R used in prototype realisation.

Algorithm 1: Selection procedure of CB@R used in the planner model.

```
1:  procedure SELECTBESTADAPTATIONOPTION
2:      bestAdaptationOption ← EMPTY
3:      bestVFC ← −MAXVALUE
4:      for each x ∈ Knowledge.AdaptationOptions do
5:          benefit ← calculateBenefit(x)                ▷ Knowledge
6:          cost ← calculateCost(x) + 1                  ▷ Knowledge
7:          VFC ← benefit/cost
8:          if VFC > bestVFC then
9:              bestVFC ← VFC
10:             bestAdaptationOption ← x
11:         end if
12:     end for
13: end procedure
```

value for the average energy consumption of the reference approach is 17.07mC, while the value for the rule-based approach is 13.10mC and for CB@R 12.98mC. Clearly, both adaptation approaches outperform the reference approach (p-value $2.2e^{-16}$ Wilcox rangsum test). Hence, we can accept hypothesis *H1*.

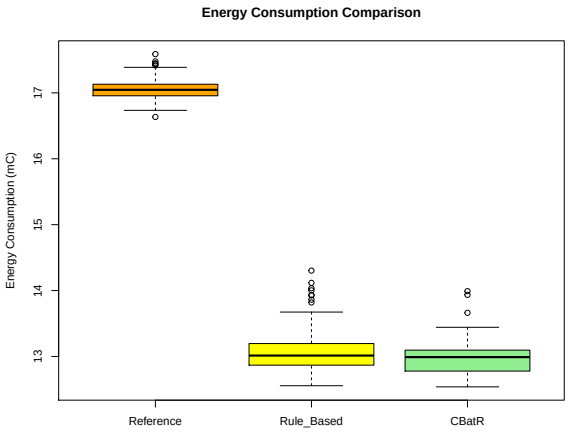

Figure 11: Evaluation results for energy consumption.

The mean values for energy consumption of the rule based approach and CB@R are similar. Although there is a significant difference of the data distribution in favour of the rule-based approach (p-value 0.044 with Wilcox rangsum test), from a practical point of view the difference is irrelevant (the relative gain in energy consumption of rule-based is versus CB@R is 0.8%). Hence, we have to reject hypothesis *H31*.

Let us now look at the packet loss. Figure 12 shows the results. The mean value for the average packet loss for the reference approach is 5,39%. The mean value for the rule-based approach is 15.32% and for CB@R it is 9.96%. Given that the average packet loss should be below 10%, we have to reject hypothesis *H21* (the rule-based approach achieves the packet loss requirement) and we can accept hypothesis *H22*. With a significant difference between the results for both approaches (p-value $6.258e^{-6}$ with Wilcox rangsum test), we can also accept hypothesis *H32* (packet

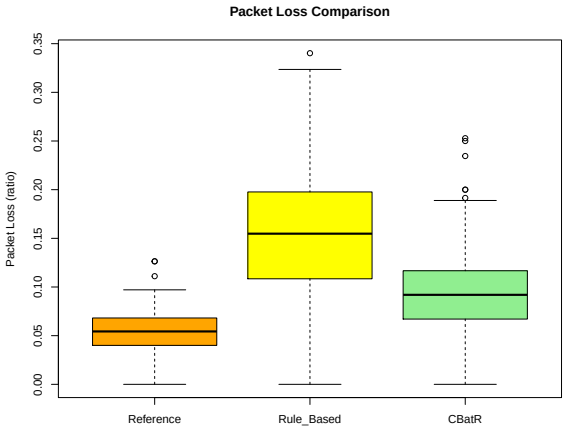

Figure 12: Evaluation results for packet loss.

loss for CB@R is significantly better as for the rule-based approach).

Discussion. For energy consumption, both self-adaptive approaches are significantly better as the reference approach. On the other hand, the rule-based approach is not able to realise the packet-loss requirement, while CB@R achieves the requirement although with a small surplus. These results show that substantial benefits can be gained by applying CB@R compared to the conservative approach. Note that differentiations to the weights for the quality properties and adjustments to the quality response curve models may help to further enhance CB@R and help improving the results for packet loss, possibly at a small extra cost of energy consumption.

Feedback from the VersaSense staff confirmed that the problem of finding and maintaining an optimal configuration of low-power wireless networks for a given problem domain is notoriously difficult and that a self-adaptive solution that automates the management of the network is a major benefit. On the other hand, the staff highlights the potential risks of automation in terms of ensuring stability during the adaptation process as well as handling outlier events. The staff also noticed the need for domain-specific models, including for different quality properties, that may not be easy to design. However, building up experience over time can mitigate such risks.

6 RELATED WORK

Until recently, most research on self-adaptation has largely ignored the negative implications of adaptation as documented by surveys, see for example (Weyns and Ahmad, 2013). In this section, we point to work that is related to the CB@R approach. We structure the discussion of related work in two parts. We start with providing brief descriptions of related approaches. Then we position CB@R in the landscape of existing work.

Bertolli et al. define a cost model for quantifying the overhead introduced by autonomic behaviour in an adaptive parallel application, in particular a flood emergency management (Bertolli et al., 2010). The application is realised using components that are specified in multiple versions. Each component is dynamically selected according to an adaptation strategy to achieve the required quality of service levels. The adaptation cost model is used to estimate the overhead for reconfiguring a component when switching between different versions.

Camara et al. propose a latency-aware adaptation approach focusing on the interval between the time when an adaptation is applied and the time when the effects of the adaptation are observed (Cámara et al., 2016). At design time, the approach uses model checking of Stochastic Multiplayer Games models to quantify the maximum improvement a latency-aware strategy is able to obtain. At runtime, a latency-aware adaptation algorithm is applied that uses simulation to support the adaptation decisions.

Cailliau and van Lamsweerde apply tradeoff analysis in an obstacle-driven runtime adaptation approach to increase the satisfaction rate of probabilistic system goals (Cailliau and van Lamsweerde, 2017). The approach uses obstacle/goal refinement trees. Leaf obstacles are monitored at runtime to determine the satisfaction rate of high-level goals. When the satisfaction rate is below a threshold, a tradeoff analysis guides the selection of alternatives to maximise satisfaction rates under cost constraints.

Poladian et al. propose ‘anticipatory configuration” to predict future resource availability to improve the utility of concurrent adaptive applications (Poladian et al., 2007). The approach is compared with reactive configuration that reacts to changes in resource availability as they occur. The results demonstrate that anticipatory configuration provides better utility to users when there is a cost (e.g., reconfiguration cost) associated to certain adaptation operations.

There are a number of related approaches that use elastic controllers in cloud-based applications to adjust the allocation of resources with awareness of the cost of adaptation. Jamshidi et al. use an elastic controller to handle unpredictable workloads in cloud applications (Jamshidi et al., 2014). The elastic controller aims at reducing the cost of ownership of the service level agreements, while auto-scaling the dynamic resources. The approach uses fuzzy logic to specify elasticity rules and handle conflicting requirements.

Gambi et al. propose a framework that explicitly takes into account the time each control action takes to complete, called actuation delay (Gambi et al., 2013). The proposed framework estimates actuation delay using change point detection algorithms. Fokaefs et al. use a business ecosystem of cloud computing as an economy of scale in a set of experiments (Fokaefs et al., 2016). The goal is to take into the account cost of infrastructure with revenue from service delivery and the profit of the service provider. The experiment results show that the economic-driven resource scalability is applicable on real cloud environments with satisfying results during elasticity operations. Finally, Jung et al. present a holistic controller framework called Mistral that uses a utility-based model to predict power consumption, cost of the adaptation, and workload prediction to maximise overall utility in cloud applications (Jung et al., 2010).

In summary, a variety of approaches have been devised to deal with the implications and costs of self-adaptation. Contrary to existing approaches that do not clearly distinguish between costs and benefits, CB@R treats cost and benefits as first-class citizens in the decision-making of adaptation. Furthermore, most of the existing approaches focus on specific types of cost for self-adaptation, ranging from overhead, latency, required resources, time for adaptation, and cost of infrastructure. CB@R does not focus on any particular type of cost, but provides a reusable framework that considers the cost for adaptation as a first-class concern that can be instantiated for the domain and problem at hand.

7 CONCLUSIONS AND FUTURE WORK

Applying self-adaptation may incur costs, which are often ignored. We applied a state-of-the-art approach for self-adaptation to a community exemplar showing that the cost for realising adaptation can be significant. This observation demonstrates that it is important for any self-adaptation solution to reflect on the cost for realising adaptation and take this into account when the cost is significant.

This paper presents CcB@R − Cost-Benefit @ Runtime − a reusable approach for decision-making in self-adaptive systems that considers both the benefits of adaptation and the cost to realise the adaptation actions as first-class citizens. The approach takes inspiration from CBAM, an established approach to deal with the economics of benefits and costs in architectural design. CB@R defines the benefits of adaptation options as the weighted utility of the qualities that the options can provide to the stakeholders. The approach defines cost as a domain-specific property that needs to be defined for the problem at hand.

We applied CB@R to a real world deployment of an IoT application. The results show that CB@R outperforms a conservative approach that is currently used by an industry partner. The results also show that contrary to a rule-based approach, CB@R achieves the adaptation goals under various uncertainties.

In future work, we plan to study different topics on cost-benefit analysis for self-adaptive systems. We plan to study the influence of varying the utility response curve models as well as the weights assigned to the different quality properties on the Value For Cost. We plan to look into methods that allow to fine tune the curves and the weights automatically using learning algorithms combined with direct or indirect feedback from stakeholders. We plan to study the scalability of the approach in terms of the number of adaptation options and adaptation goals that can be handled. In this context, we plan to study online learning techniques to select adaptation options from large sets. As for the cost of adaptation, we plan to study more systematically the types of costs that apply to self-adaptation, derive a classification for these costs, and apply this knowledge to define methods to support different types of costs in self-adaptation.

REFERENCES

Bennaceur, A., McCormick, C., García-Galán, J., Perera, C., Smith, A., Zisman, A., and Nuseibeh, B. (2016). Feed me, feed me: An exemplar for engineering adaptive software. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2016 IEEE/ACM 11th International Symposium on*. IEEE.

Bertolli, C., Mencagli, G., and Vanneschi, M. (2010). A cost model for autonomic reconfigurations in high-performance pervasive applications. In *Proceedings of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems*, page 3. ACM.

Blair, G., Bencomo, N., and France, R. B. (2009). Models@ run. time. *Computer*, 42(10).

Cailliau, A. and van Lamsweerde, A. (2017). Runtime monitoring and resolution of probabilistic obstacles to system goals. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 1–11.

Cámara, J., Moreno, G. A., Garlan, D., and Schmerl, B. (2016). Analyzing latency-aware self-adaptation using stochastic games and simulations. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(4):23.

CBAM (2018). Cost Benefit Analysis Method, SEI, CMU https://www.sei.cmu.edu/architecture/tools/evaluate/cbam.cfm.

Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., and Whittle, J. (2009). *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Springer Berlin Heidelberg, Lecture Notes in Computer Science vol. 5525.

de Lemos, R., Garlan, D., Ghezzi, C., Giese, H., Andersson, J., Litoiu, M., Schmerl, B., Weyns, D., Baresi, L., Bencomo, N., Brun, Y., Camara, J., Calinescu, R., Chohen, M., Gorla, A., Grassi, V., Grunske, L., Inverardi, P., Jezequel, J., Malek, S., Mirandola, R., Mori, M., Müller, H., Rouvoy, R., Rubira, C., Rutten, E., Shaw, M., Tamburrelli, G., Tamura, G., Villegas, N., Vogel, T., and Zambonelli, F. (2017). *Software Engineering for Self-adaptive Systems: Research Challenges in the Provision of Assurances*. Springer Berlin Heidelberg, Lecture Notes in Computer Science vol. 9640.

De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N. M., Vogel, T., et al. (2013). Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer.

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

Fokaefs, M., Barna, C., and Litoiu, M. (2016). Economics-driven resource scalability on the cloud. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM.

Gambi, A., Moldovan, D., Copil, G., Truong, H.-L., and Dustdar, S. (2013). On estimating actuation delays in elastic computing systems. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE.

Garlan, D., Cheng, S., Huang, A., Schmerl, B., and Steenkiste, P. (2004). Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*, 37(10):46–54.

Gerasimou, S., Calinescu, R., Shevtsov, S., and Weyns, D. (2017). Undersea: an exemplar for engineering self-adaptive unmanned underwater vehicles. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE Press.

Iftikhar, M. U., Ramachandran, G. S., Bollansée, P., Weyns, D., and Hughes, D. (2017). Deltaiot: a self-adaptive internet of things exemplar. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017 IEEE/ACM 12th International Symposium on*, pages 76–82. IEEE.

Iftikhar, M. U. and Weyns, D. (2014). ActivFORMS: Active formal models for self-adaptation. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM.

Iglesia, D. G. D. L. and Weyns, D. (2015). Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(3):15.

Jamshidi, P., Ahmad, A., and Pahl, C. (2014). Autonomic resource provisioning for cloud-based software. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM.

Jung, G., Hiltunen, M. A., Joshi, K. R., Schlichting, R. D., and Pu, C. (2010). Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *International Conference onDistributed Computing Systems (ICDCS)*. IEEE.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Kramer, J. and Magee, J. (2007). Self-Managed Systems: An Architectural Challenge. In *Future of Software Engineering*, (FOSE). IEEE Computer Society.

Len, B., Paul, C., and Rick, K. (2013). Software architecture in practice. *Addison-Wesley*.

Oreizy, P., Medvidovic, N., and Taylor, R. (1998). Architecture-based Runtime Software Evolution. In *International Conference on Software Engineering*, (ICSE). IEEE Computer Society.

Poladian, V., Garlan, D., Shaw, M., Satyanarayanan, M., Schmerl, B., and Sousa, J. (2007). Leveraging resource prediction for anticipatory dynamic configuration. In *International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE.

Salehie, Mazeiar, T. L. (2009). Self-adaptive software: Landscape & research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2).

Weyns, D. (2018). Software engineering of self-adaptive systems: an organised tour and future challenges. In *Software Engineering Handbook*. Springer. (Eds.) Taylor, R. and Kang, K. C. and Cha, S.

Weyns, D. and Ahmad, T. (2013). Claims and evidence for architecture-based self-adaptation: A systematic literature review. In *European Conference on Software Architecture*, pages 249–265. Springer.

Weyns, D., Iftikhar, M. U., and Söderlund, J. (2013). Do external feedback loops improve the design of self-adaptive systems? a controlled experiment. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, (SEAMS).

Weyns, D., Malek, S., and Andersson, J. (2012). FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(1):8.