Analysing and modelling runtime architectural stability for self-adaptive software

Salama, Maria; Bahsoon, Rami

DOI:
10.1016/j.jss.2017.07.041

Document Version
Peer reviewed version

Citation for published version (Harvard):
Salama, M & Bahsoon, R 2017, 'Analysing and modelling runtime architectural stability for self-adaptive software', *Journal of Systems and Software*, vol. 133, pp. 95-112. https://doi.org/10.1016/j.jss.2017.07.041

Analysing and Modelling Runtime Architectural Stability for Self-Adaptive Software

Maria Salama, Rami Bahsoon

PII: S0164-1212(17)30162-0
DOI: 10.1016/j.jss.2017.07.041
Reference: JSS 10014

To appear in: *The Journal of Systems & Software*

Received date: 27 June 2016
Revised date: 14 July 2017
Accepted date: 27 July 2017

Please cite this article as: Maria Salama, Rami Bahsoon, Analysing and Modelling Runtime Architectural Stability for Self-Adaptive Software, *The Journal of Systems & Software* (2017), doi: 10.1016/j.jss.2017.07.041

Highlights

- Analysing and modelling run-time architectural stability for self-adaptive software
- Focus on stabilising the quality of service provision and quality of adaptation
- Consider multiple stability concerns and viewpoints
- Application to the case of cloud architectures

Analysing and Modelling Runtime Architectural Stability for Self-Adaptive Software

Maria Salama[a,*], Rami Bahsoon[a]

[a]*School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK*

Abstract

With the increased dependence on software, there is a pressing need for engineering long-lived software. As architectures have a profound effect on the life-span of the software and the provisioned quality of service, stable architectures are significant assets. Architectural stability tends to reflect the success of the system in supporting continuous changes without phasing-out. For self-adaptive architectures, the *behavioural* aspect of stability is essential for seamless operation, to continuously keep the provision of quality requirements stable and prevent unnecessary adaptations that will risk degrading the system. In this paper, we introduce a systematic approach for analysing and modelling architectural stability. Specifically, we leverage architectural concerns and viewpoints to explicitly analyse stability attributes of the intended behaviour. Due to the probabilistic nature of systems' behaviour, stability modelling is based on a probabilistic relational model for knowledge representation of stability multiple viewpoints. The model, empowered by the quantitative analysis of Bayesian networks, is capable to conduct runtime inference for reasoning about stability under runtime uncertainty. To illustrate the applicability and evaluate the proposed approach, we consider the case of cloud architectures. The results show that the approach increases the efficiency of the architecture in keeping the expected behaviour stable during runtime operation.

Keywords: software architecture, architectural stability, self-adaptive architecture, sustainability, quality of service, cloud architecture

1. Introduction

Modern software systems are increasingly operating in highly open and dynamic environments [1]. Subsequently, self-adaptation has widely emerged for

*Corresponding author

Email addresses: m.salama@cs.bham.ac.uk (Maria Salama), r.bahsoon@cs.bham.ac.uk (Rami Bahsoon)

engineering modern software systems to achieve the necessary level of dynamicity and scalability [2]. For a quick response to runtime changes, systems autonomously and dynamically adapt their architectures, in order to regulate the satisfaction of functional and quality requirements [3] [4]. Self-adaptive architectures are built with self-managing and controlling capabilities following the principles of autonomic computing, to respond to changes in user requirments and the execution environment, as well as to cope with uncertainty in runtime operation [5].

As architectures have a profound effect on the life-span of the software and the quality of service (QoS) provision [6] [7], the architecture's behaviour tends to reflect the success of the system in constantly provisioning end-users' requirements, as well as supporting and tolerating continuous changes and evolution over time. We argue that architectural stability manifests itself as an architectural property necessary for the operation of self-adaptive software, their dependability and long-livety over time [8]. To leverage the capabilities of self-adaptive systems, it is necessary to consider behavioural stability to ensure that the architecture's intended behaviour is provisioned during runtime operation.

An extensive literature survey [9] has revealed that the stability property has been considered at different levels (e.g. code, whole program, design, architecture levels) and with respect to several aspects (e.g. logical, structural, physical). This implies many different interpretations for considering stability as a quality attribute. At the architecture level, stability has been viewed as the ability to endure with changes in requirements and the environment, while reducing the likelihood of architectural drifting and phasing-out, by avoiding ripple structural modifications (over two or more versions the software) [10] [11]. That is an *evolutionary perspective* in considering stability, i.e. evolving the system through a number of releases [12]. Meanwhile, dynamic changes, which occur while the system is in operation, require quick and dynamic adaptations during runtime [12]. This calls for an *operational perspective* of stability that is fundamental for self-adaptive software architectures, to ensure seamless operation.

Even though adaptation mechanisms have been widely investigated in the engineering of dynamic software systems, stability was not explicitly tackled [13]. The shortcoming of current software engineering practice regarding stability is that the stable provision of certain quality attributes essential for end-users (e.g. response time for real-time systems) is not explicitly considered in the adaptation decision taken during runtime. Besides, the adaptation process does not address the adaptation properties that affect the quality of adaptation. An adaptation indefinitely repeating the action or making frequent adaptations will risk not improving or even degrading the system's behaviour to unacceptable levels [13] [14]. The challenge we address in this paper is *how to systematically handle architectural stability as a behavioural aspect during the runtime operation of the system*, so that the system can be seamlessly adapted and ensure a constant provision of the intended services. A stable self-adaptive architecture is expected to keep the fulfilment of QoS objectives stable, while performing adaptations that converge towards these objectives and eliminating unnecessary

ones.

To address this challenge, we propose a new systematic approach for analysing and modelling architectural stability, focusing on the behavioural aspect during runtime. The analysis model aims to capture stability dimensions, stakeholders' concerns for stability and related attributes. Given the non-deterministic behaviour of the systems, modelling stability is based on probabilistic relational model for knowledge representation of stability multiple viewpoints and related attributes. The mathematical model empowers the quantitative analysis of Bayesian networks for modelling dynamic impact and correlation assessment among stability attributes and analysing associated trade-offs. This approach can effectively conduct runtime inference to reason about stability attributes given the continuous runtime uncertain changes. Such reasoning improves the quality of adaptation for achieving the intended behaviour and supporting seamless operation. The approach for considering stability shall be an integral part of self-adaptive software systems runtime operation to ensure effectiveness and long-term use.

The main contributions of our work are as follows.

- We propose a model for analysing stability based on architectural concerns and viewpoints. Stability viewpoints frame the stakeholders' concerns for the system's behaviour along with the dimensions of stability that reflect the architecture type. Stability attributes are, then, defined to present the details of the intended behaviour needed to be kept stable.

- We mathematically model the non-deterministic behaviour of stability attributes using probabilistic modelling. We present the interdependencies between stability attributes using probabilistic relational model. Based on that, we quantitatively measure the strengths of dependence relations and sensitivity among stability attributes, and construct the Bayesian network using observed data. With the help of Bayesian networks, we conduct runtime inference to measure the probable effect of stability attributes for reasoning about the whole architecture's behaviour under runtime uncertainty.

- We introduce a systematic approach for considering stability as an architectural property. The approach consists of three subsequent main phases: (i) stability analysis aims at building the *stability qualitative model* that analyses and presents the intended behaviour, (ii) stability modelling captures the probabilistic relation between interdependent stability attributes by building the *stability quantitative model*, and (iii) runtime support which employs the model for runtime inference and reasoning about stability under runtime uncertainty.

- We apply the proposed approach on the case of self-adaptive cloud architectures. The analysis model has shown promising capability in exploring dimensions, concerns and attributes related to stability, and hence, drawing a comprehensive and explicit consideration of stability as an architectural property. The probabilistic model has quantitatively captured the

impacts and correlations between stability attributes. We conduct experimental evaluation using the *RUBiS* benchmark [15] and the *World Cup 1998* workload trend [16], to stress the architecture with variations of runtime changing workloads. The results show that reasoning about stability using the the runtime inference has improved the adaptation decision and achieved the intended behavioural requirements with less violations.

Organisation. In section 2, we describe the background and discuss related work. Section 3 and 4 elaborates the technical contributions of behavioural stability analysis and modelling. Section 5 presents our holistic approach for supporting runtime behavioural stability. Section 6 applies our approach to the case study, followed by the experimental evaluation in Section 7 and the discussion in Section 8. We discuss the threats to validity of the proposed work in section 9. Section 10 concludes the paper and indicates future work.

2. Background

In this section, we introduce the main concepts (sub-section 2.1), sketch the properties of architectural stability as a quality attribute (sub-section 2.2), and discuss related work (sub-section 2.3).

2.1. Definitions of the Main Concepts

Software Architecture. The concept of software architecture has been defined in different ways under different contexts. In our work, we adopt the definition of the ISO/IEC/IEEE Standards that defines software architecture as the "fundamental organisation of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [17]. This definition is in line with early definitions when the discipline has emerging [18] [19] and with matured ones appearing later [20]. Software architectures provide abstractions for representing the structure, behaviour and key properties of a software system [19]. They are described in terms of software components (computational elements), connectors (interaction elements), their configurations (specific compositions of components and connectors) and their relationship to the environment [21] [22].

Software life cycle. The life cycle of a software system consists basically of the *development* and *operation* phases [23]. The development phase includes all activities till the decision that the software is ready for operation to deliver service, such as requirements elicitation, conceptual design, architectural design, implementation and testing [23]. The operation phase begins when the system is deployed, configured and put into operation to start delivering the actual service in the end-user's environment, cutover issues are resolved and the product is launched [23] [17]. The former phase is known as *initial development* or *design-time*, and the latter is usually referred as *runtime*. After the development and launch of the first functioning version, the software product enters to different

cycles of maintenance and evolution stages till reaching the phase-out and close-down [24] [23] [17]. During the maintenance stage, minor defects are repaired, while the system functionalities and capabilities are extended in major ways in the evolution stage [24].

Quality Attribute. The definition of a quality attribute we use is of the IEEE Standard for a Software Quality Metrics defining quality attribute as "a characteristic of software, or a generic term applying to quality factors, quality subfactors, or metric values" [25]. According to the same standard, a *quality requirement* is defined as "a requirement that a software attribute be present in software to satisfy a contract, standard, specification, or other formally imposed document" [25].

Architecturally-significant requirements. Generally, the architecture should fulfil the software requirements, both functional requirements (what the software has to do) and quality requirements (how well the software should perform) [26] [27]. Functional requirements are implemented by the individual components, while the quality requirements are highly dependent on the organisation and communication of these components [28]. In the software architecture discipline, the architecturally-significant requirements are considered, as not all requirements have equal effect on the architecture [29]. Architecturally-significant requirements are a subset of technically challenging requirements, technically constraining and central to the system's purpose. These requirements have signficant influence on the architecture design decisions, as they should be satisfied by the architecture [29]. *Architecturally-significant functional requirements* may define the essence of the functional behaviour of the system [30], while *architecturally-significant quality requirements* are often technical in nature, such as performance targets [31] [32]. This special category of requirements, describing the key behaviours that the system should perform, plays a main role in making architectural decisions and has measurable effect on the software architecture.

System Behaviour. The behaviour of a system is the "observable activity of the system, measurable in terms of quantifiable effects on the environment whether arising from internal or external stimulus" [17]. This is determined by the state-changing operations the system can perform [17].

Self-adaptive software system. In general settings, *to adapt* means "to change a behaviour to conform to new circumstances" [33]. A self-adaptive software "evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible" [34] [5] [3]. Intuitively, a self-adaptive system is one that has the capability of modifying its behaviour at runtime in response to changes in the dynamics of the environment (e.g. workload) and disturbances to achieve its goals (e.g. quality requirements) [35]. Self-adaptive systems are composed of two sub-systems: (i) the managed system (i.e. the system to be controlled), and (ii) the adaptation controller (the managing system)

[13]. The managed system structure could be either a non-modifiable structure or modifiable structure with/without reflection capabilities (e.g. reconfigurable software components architecture) [13]. The controller's structure is a variation of the MAPE-K loop [13].

2.2. Architectural Stability

Generally, the notion of "stability" refers to the resistance to change and the tendency to recover from perturbations. The condition of being stable, thus, implies that certain properties of interest do not (very often) change relative to other things that are dynamically changing. Stability has been defined in different domains and disciplines, such as nature, ecology, chemistry and mathematics.

As a software quality property, stability is defined in the ISO/IEC 9126 standards for software quality model [36] as one of the sub-characteristics of the maintainability characteristic of the software, along with analysability, changeability and testability. Maintainability is "the capability of the software product to be modified in order to cope with changes in requirements and environment or to handle errors" [36]. Stability itself is "the capability of the software product to avoid unexpected effects from modifications of the software" [36]. For general application purposes, the standard does not determine specific features or aspects for stability [37].

Reviewing the state-of-the-art in software engineering [9], we have found that stability has been considered at different levels, i.e. at the code level (e.g. [38]), requirements (e.g. [39]), design ([40] [41] [42] [43] [44]) and at the architecture level ([45] [10] [46] [47]). At each level, stability has been considered in relation to several aspects from different perspectives, and thus interpreted in many ways according to the perspective of consideration For instance, stability at the code level has been interpreted as "the resistance to the potential ripple effect that the program would have when it is modified" [38], that is considering the *logical* and performance (i.e. *behavioural*) aspects of stability from the *maintenance* perspective. Design stability has been refered to "the extent to which the structure of the design is preserved throughout the evolution of the software from one release to the next" [40], where the *logical* and *structural* aspects of stability are considered from *evolutionary* perspective.

Architectural stability has been considered in terms of ripple structural modifications over two or more versions of the software, as a *structural* aspect with respect to architecturally-relevant changes carried from *evolutionary* ([45] [10]) and maintenance perspectives ([47]). This has been referred to the extent to which the architecture's structure is capable to accommodate the evolutionary changes without re-designing the architecture or making ripple modifications [45] [10]. For the different perspectives, the *structural* aspect of stability is the one mostly considered at the architecture level.

Considering self-adaptive software systems, the structure and the behaviour of the software may be affected when adaptations are taking place during runtime [48]. In this context, we distinguish between the *structural* and *behavioural* aspects of stability. We also posit that an *operational* perspective (during the

runtime operation of the software) for stability is essential for self-adaptive systems, different from the *evolutionary* perspective (over two or more versions of the software). The stability meaning, we are seeking, can be regarded at the *architectural* level as a *runtime* property considering the *behavioural* aspect from an *operational* perspective.

Inspired by stability studies in the "Control Theory" discipline [49] [50] - that has been widely used to incorporate self-adaptive capabilities into software systems [51], we posit that stability occupies a key position for the reason that the upper limit of the performance of the architecture is often set by stability considerations [50]. A stable architecture is an architecture that, "when perturbed from an equilibrium state, will tend to return to that equilibrium state" [50]. So, stability of the architecture is essential to examine the behaviour with time following a perturbation during runtime.

Given the different levels, aspects and perspectives of considering stability, we view that a precise definition should focus on a simple ability (e.g. ability to keep unchanged or recover from perturbations) based on the intended perspective (e.g. evolutionary or operational), as well as a specific level and aspect. For instance, one possible definition could be *the ability of the architecture's structure to keep unchanged along with the time to endure evolutionary changes*. Such definition targets stability at the architecture level from an evolutionary perspective and focuses on the structural aspect. Another possible definition could be *the ability of the architecture's behaviour to maintain a fixed level of operation (or recover from operational perturbations) within specified tolerances under varying external conditions* for considering the *behavioural* aspect from an *operational* perspective. By that, a stable architecture from the *operational* perspective is the one capable to continuously fulfil the architecturally-significant quality requirements during runtime, where the architecture can return to the equilibrium state, following perturbation due to changes in quality requirements, workload patterns or in the operational environment. Conversely, an unstable architecture is one that, when perturbed from equilibrium, will show deviation from the expected behaviour.

2.3. Related Work

The *runtime behavioural stability of software architectures* was not explicitly tackled as an architectural property in the literature to date, to the best of our knowledge. The work of Gorbenko et al. [52] could be considered as a partial exception. This study investigated the instability of service-oriented architectures, focusing on the instability of three attributes: performance, response time and communication delays. Though, this work could be considered partially tackling the stability of the architecture's behaviour (performance characteristics) during runtime, the explicit focus of this work was on dependability and resilience, not explicitly considering stability as an architectural property.

The *architecture analysis* community has developed mehtods for predicting the quality provision of architecture design alternatives during design-time [37]. Examples include Scenario-based Architecture Analysis Method (SAAM) [53],

Architecture Tradeoff Analysis Method (ATAM) [54] and quality impact analysis [55] which focused on traditional quality attributes, not their stability. Other studies focused on estimating system failures or predicting the probability that the system will perform its intended functionality aiming at reducing or eliminating failures [56] [57] [58] [59]. Architecture analysis methods cannot be used to support the runtime provision of quality requirements and their stability, given the uncertainty during operation and the automation and quantitative analysis required for runtime operation.

Considering self-adaptive architectures, the *adaptation mechanisms* proposed in the literature focused on some adaptation properties, such as tactics latency (the time it takes since an adaptation is started until its effect is observed) [60], settling time (the amount of time the controller takes to achieve the adaptation goal) [61] [62]. Yet, properties reflecting the quality of adaptation, i.e. how well the adaptation process converges towards the adaptation objective, are not explicitly considered [63] [13]. Meanwhile, properties reflecting the behaviour of the controller have impact on the stability of the whole architecture [13].

3. Stability Analysis

Architectural stability could not be considered an absolute action, or rather it is relative to the type of the architecture and its intended behaviour. In particular, the architecture type (i.e. self-adaptive) and the application domain (e.g. mobile-, web-, cloud-based) have direct inputs to behavioural stability. As an example of behaviour, one architecture could be intended to keep the response time stable (as it is a crucial quality attribute for the end-users in the case of real-time systems), while energy consumption could be a critical requirement attribute to be kept stable for another architecture. We argue that stability is a relative matter subject to the concerns of stability and the type of the architecture. Thus, stability should be considered relatively to these concerns. This calls for more expressive abstractions to represent the concerns and their related attributes subject of stability. The analysis aims to capture the relevant attributes that characterise stability concerns and stability dimensions, as well as their influence on each other's stability.

To consider the architecture type in the analysis, we view two main *stability dimensions* for self-adaptive software architectures, that are *adaptation goals* and *adaptation properties*. Both underlies the functioning of a self-adaptive system, that we intend to stabilise its architectural behaviour (i.e. the managed system and the managing system) [13]. Adaptation goals are the quality of service (QoS) properties intended to be achieved by the architecture, while adaptation properties are observed and measured in the adaptation process [13]. These two distinct dimensions allows considering both the quality requirements and the behaviour of the adaptation controller in the analysis of behavioural stability of the architecture as a whole.

For analysing stability, we exploit one of the holistic reasoning methods for quality analysis in software architectures. In particular, we extend the

"ISO/IEC/IEEE 42010 Systems and software engineering - Architecture description" standards [64], as outlined in Figure 1). [1]

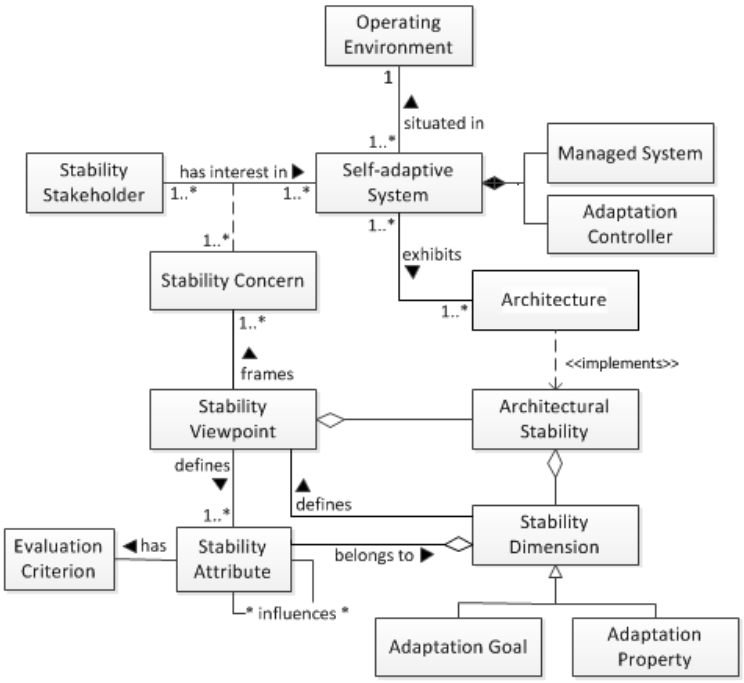


Figure 1: Architectural Stability Analysis Model.

According to the ISO/IEC/IEEE 42010 [64], a system has one or more stakeholders, where each stakeholder has interest (i.e. concerns) for that system. Concerns are "those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders" [64]. Examples of concerns include quality of service, enviromental regulations and economical concerns. We envision mapping the stability analysis to the well-known architecture related concept "architectural concerns" that refer to the requirements of different stakeholders [64]. Considering stability, stakeholders' concerns for stabilising the architecture behaviour can be seen as architectural concerns or *stability concerns*.

Having different stakeholders, viewpoints have been introduced to support the modelling, understanding and analysis of software architectures for different stakeholders [65], delineating the architectural information that address stakeholders' concerns [66]. Architectural viewpoints refer to the conventions for

[1] The figure uses UML notation of ISO/IEC 19501:2005, Information Technology Open distributed processing Unified modeling language (UML) Version 1.4.2.

constructing and using architectural representation addressing the requirements of different stakeholders [64] [67] [68]. Analysing stability from different perspectives can be seen as architectural viewpoints or *stability viewpoints*. We consider stability viewpoints as a model for framing stakeholders' concerns and representing architectural stability from different perspectives.

Realising runtime behavioural stability requires continuous provision of quality requirements. Following the approach of well-established architectural methods, which considers quality attributes [31] [54] [69] as the base for architectural analysis, we analyse stability in relation to the attributes that are required to be kept stable throughout the operation of the architecture, i.e. *stability attributes*. Attributes that are subject to stability are defined for different viewpoints reflecting stakeholders' concerns, including traditional quality of service attributes, which are the adaptation goals [13]). Since adaptations are motivated by the need of continued satisfaction of quality requirements, the analysis should also consider attributes of the adaptation properties [13], in order to reflect how adaptations converge towards adaptation goals.

Stability attributes are interdependent, i.e. may influence each other, either by supporting or by contradicting each other. So, architects should, for explicitly targeting stability, analyse the interdependency and correlation between different stability attributes (appearing as *influences* relation between stability attributes in Figure 1) and resolve related trade-offs.

While traditional architecture analysis considers dependencies and trade-offs analysis between traditional quality attributes, such as performance and availability [70], stability analysis involves multiple viewpoints and related attributes, as well as analyses their interdependencies and trade-offs. These include not only traditional quality attributes, but also adaptation properties that affect the architecture's behaviour for continuously satisfying quality requirements. Using the analysis model for identifying stability viewpoints, attributes and their dependencies explicit would help architects appreciate behavioural stability beyond traditional quality attributes.

4. Stability Modelling

4.1. Stability Modelling

Achieving runtime architectural stability for different viewpoints should involve a careful understanding of the relationship, impact, correlation and sensitivity among stability attributes, as well as handling potential conflicts between different viewpoints. Such attributes are non-deterministic given the uncertainty associated with the runtime operation. Uncertainty, affecting the architecture's operation, can be attributed to many facets, such as changes in workload, quality requirements, runtime goals, and the environment where the architecture is operating [71] [72]. Therefore, probabilistic modelling is appropriate for modelling stability given the runtime operational uncertainties, since deterministic analysis is limited when dealing with such operational uncertainties.

It is possible for the architect to use alternative techniques to Bayesian network, where the architect can evaluate the extent to which changes to one or

more attributes value can potentially influence the stability of the architecture. Asserting changes to attribute value can be reliable when informed by expert judgement or accompanied with careful assessment of the application domain. One potential problem, however, is that the analysis tend to be human-centred, subjective and can miss potential cases that are change-revealing, as such techniques rely on human judgement and sensitivity analysis. Furthermore, the analysis can be difficult to scale and handle in cases where more than one attribute can potentially change or higher number of attributes are under evaluation. It worth noting that our method can complement existing architecture analysis and evaluation methods (e.g. Architecture Tradeoff Analysis Method (ATAM) [54] and quality impact analysis [55]) to provide automatic and probabilistic assessment, which replace and improve human assertions for attribute value and its likely influence on the trade-offs analysis and the choice of decisions. Probabilistic assessment is especially important for architectures that exhibit high degree of uncertainty in their operation which is the case of self-adaptive systems.

Meanwhile, we adhere to the Bayesian choice in the automated reasons about stability during runtime for many reasons. As a consistent and complete representational tool, it is guaranteed to define a unique probability distribution over the network variables [73]. Also, the the Bayesian network is a compact representation, as it allows one to specify an exponentially sized probability distribution using a polynomial number of probabilities [73]. The coherence of the Bayesian statistical inference is another important feature. By modelling the unknown parameters of the sampling distribution through a probability structure, i.e. by probabilising uncertainty, the Bayesian approach authorises a quantitative discourse on these parameters [74]. The Bayesian approach is also known to be the only system allowing for conditioning on the observations, effectively implementing the Likelihood Principle and frequented optimality notions of Decision Theory [74].

Probabilistic modelling consists of two components: (i) the structure, often referred as the qualitative model, and (ii) the parameters (i.e. conditional probabilities) referred as the quantitative model [75]. For the former, we use Probabilistic Relational Models that are able to harness the expressive power of architecture analysis. For the latter, Bayesian networks feature the ability to quantitatively perform dynamic impact analysis and correlation assessment among stability attributes under runtime uncertainty [70]. Generally, Bayesian networks have proven to be ideally suited knowledge representations for reasoning and decision making under uncertainty [75], i.e. reasoning over probabilistic causal models under uncertainty [76]. Bayesian networks have been widely used for the modelling and analysis of uncertain phenomena which are known to be causally connected [77]. With the capability of representing probabilistic behaviours in a compact and intuitive way [56], Bayesian networks are applicable for domain areas with inherent uncertainty [75], which is applicable to the case of architecture's behaviour at runtime.

We view stability attributes as the "knowledge" to be presented by Probabilistic Relational Models, as these attributes tend to vary during runtime.

Probabilistic modelling, empowered by the quantitative analysis of Bayesian networks, aims to model the wide variations of probable values linked to stability attributes that we are interested in, as well as understand their likely ramifications on other attributes and their trade-offs under runtime uncertainty.

Our approach for modelling stability follows the formalism process of probabilistic relational models [75], that is suitable for representing and processing probabilistic knowledge of runtime behavioural stability. For each viewpoint, a probabilistic relational model is constructed using the stability attributes identified earlier in the analysis. The model represents the relation between the attributes of the viewpoint and interdependent attributes. The approach for eliciting the model structure relies on the notion of cause-effect relations between the variables of the problem domain [75]. In practice, such relations are modelled using a graph of nodes representing the variables and links representing the cause-effect relations between the entities.

The construction of probabilistic networks usually proceeds according to an iterative procedure, where the set of nodes and the set of links are updated iteratively as the model becomes more and more refined [75]. Modelling causal dependence relations requires careful consideration, as sometimes it is not quite obvious in which direction a link should point [75]. In the case of architectural stability, we can rely on the architect's experience, subject matter experts and pre-experiments in defining the dependency relations between different stability attributes. Structure learning could also make use of data-driven approaches, where data could be acquired from pre-experiments and simulations. There exist different classes of algorithms for learning the structure of Bayesian networks, such as search-and-score and constraint-based algorithms [75]. Background knowledge of domain experts and architects can be specified in the form of constraints on the structure of the model.

Having the probabilistic relational model established, this defines the structure of the Bayesian network, where the elicitation of the quantitative information will take place. We use Bayesian networks to model the dynamic non-deterministic behaviour of stability attributes, that change with a range of values at runtime and tend to interfere among each other, collectively influencing the behaviour of the architecture.

A Bayesian network is a directed acyclic graph (DAG), where the nodes represent stochastic uncertain variables [78] [56], which are the stability attributes in our case. The edges of the graph are the dependencies between the nodes, showing influential relations between the variables [78] [56]. The nodes' dependencies are specified qualitatively by the edges and quantitatively by the *conditional probability distributions*. The underlying joint probability is decomposed as a product of *local conditional probability distributions* (CPDs) associated with each node and its respective parents. The CPDs are represented as *node probability tables* (NPTs), which list the probability that the child node takes on each of its different values for each combination of values of related nodes.

Formally (following [79] [75]), a discrete Bayesian network $\mathcal{N} = (\mathcal{X}, \mathcal{G}, \mathcal{P})$ consists of a set of n discrete random variables (stability attributes) $\mathcal{X}$, a directed

acyclic graph $\mathcal{G} = (\mathsf{V}, \mathsf{E})$, and a set of conditional probability distributions $\mathcal{P}$. Each variable $\mathsf{X}_i \in \mathcal{X}, \quad 1 \leq i \leq n$ is represented by a node v_i of $\mathcal{G}$ and has a finite set of mutually exclusive states $\mathsf{dom}(\mathsf{X}_i)$. The directed edges E of $\mathcal{G}$ specify assumptions of conditional dependencies between the nodes, where a directed edge from X_i to X_j is in E iif X_i is a parent of X_j. Each variable $\mathsf{X}_i \in \mathcal{X}$ has a conditional probability distribution $\mathsf{P}(\mathsf{X}_i|\mathsf{X}_{pa(v_i)}) \in \mathcal{P}$, that specifies the probabilistic dependence between the node v_i and its parents $pa(v_i) \in \mathsf{V}$.

Definition. A discrete Bayesian network $\mathcal{N} = (\mathcal{X}, \mathcal{G}, \mathcal{P})$ consists of

- a set of discrete random variables $\mathcal{X} = \{\mathsf{X}_1, ..., \mathsf{X}_n\}$
- a directed acyclic graph $\mathcal{G} = (\mathsf{V}, \mathsf{E})$ with nodes $\mathsf{V} = \{v_1, ..., v_n\}$ representing the variables of $\mathcal{X}$ and directed edges $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$
- a set of conditional probability distributions $\mathcal{P}$ containing probability distribution $\mathsf{P}(\mathsf{X}_v|\mathsf{X}_{pa(v)})$ for each variable $\mathsf{X}_v \in \mathcal{X}$

The joint probability distribution of the Bayesian network $\mathcal{N}$ is obtained by the multiplicative factorisation of the joint probability distributions $\mathcal{P}$ over the set of variables $\mathcal{X}$ as represented by the chain rule of Bayesian networks:

$$\mathsf{P}(\mathcal{X}) = \prod_{v \in \mathsf{V}} \mathsf{P}(\mathsf{X}_v|\mathsf{X}_{pa(v)}) \tag{1}$$

The Bayesian network is constructed by computing *prior* probabilities, i.e. $\mathsf{P}(\mathcal{X})$ for all $\mathsf{X} \in \mathcal{X}$, collected from empirical data in order to get initial probability values.

Capturing dependency factors between stability attributes, the constructed Bayesian network for stability provides a powerful tool for reasoning and decision support, as it can be used to reason about the effect of stabilising a specific attribute on the stability of other attributes. By that, an adaptation action achieving the stability of the whole architecture's intended behaviour could be derived for multiple stability concerns, viewpoints and attributes. Also, behavioural stability could be estimated under changing runtime workloads.

4.2. Stability Runtime Inference

The Bayesian network model representation of a problem domain can be used as the basis for drawing inference and performing analysis about the domain, in order to support reasoning under uncertainty. Decision options and utilities associated with these options can be incorporated explicitly into the model, where the model becomes capable of computing expected utilities of all decision options given the information known [75]. Since a Bayesian network encodes all relevant qualitative and quantitative information contained in a full probability model, it is a well-suited tool for many types of probabilistic inference.

The Bayesian model is used to support reasoning about stability under runtime uncertainty, which requires dynamically computing the probability states of stability attributes given the runtime changes. That is the task of computing

the *posterior* probability distribution of some variables of interest conditioned on some other variables that have been observed [75].

A Bayesian inference approach starts with the *priori* knowledge about the model structure. This initial knowledge, represented in the form of prior probability distribution gathered during the construction of the model, is updated to obtain *posterior* probability distribution over the model. By observing which states the nodes of the Bayesian network assume, known as *events*, we obtain the evidence ε for a subset of these nodes. With the help of evidence, we can compute the *posterior* marginals given a set of evidence ε, which are $\mathsf{P}(\mathsf{X}|\varepsilon)$ for all $\mathsf{X} \in \mathcal{X}$. If the evidence set is empty $\varepsilon = \phi$, then the task is to compute all prior marginals, i.e. $\mathsf{P}(\mathsf{X})$ for all $\mathsf{X} \in \mathcal{X}$.

Exploiting the independence relations induced by the structure of $\mathcal{G}$ and the evidence, let us consider the general case of computing the posterior marginal $\mathsf{P}(\mathsf{X}_i|\varepsilon)$ of a variable X_i given evidence ε. Let $\varepsilon = \{\varepsilon_1, ..., \varepsilon_m\}$ be a non-empty set of evidence over variables $\mathcal{X}(\varepsilon)$. For a non-observed variable $\mathsf{X}_j \in \mathcal{X}$, the task to compute the posterior probability distribution $\mathsf{P}(\mathsf{X}_j|\varepsilon)$ can be done by exploiting the chain rule factorisation of the joint probability distribution (equation 1):

$$\mathsf{P}(\mathsf{X}_j|\varepsilon) = \frac{\mathsf{P}(\varepsilon|\mathsf{X}_j)\mathsf{P}(\mathsf{X}_j)}{\mathsf{P}(\varepsilon)}$$

$$= \sum_{\mathsf{X} \in \mathcal{X} \setminus \{\mathsf{X}_j\}} \prod_{\mathsf{X}_i \in \mathcal{X}} \mathsf{P}(\mathsf{X}_i|\mathsf{X}_{pa(v_i)}) \prod_{\mathsf{X} \in \mathcal{X}(\varepsilon)} \varepsilon_{\mathsf{X}} \tag{2}$$

for each $\mathsf{X}_j \notin \mathcal{X}(\varepsilon)$, where ε_{X} is the evidence function for $\mathsf{X} \in \mathcal{X}(\varepsilon)$ and v_i is the node representing X_i. By that, we can observe the state of all stability attributes, and hence the stability state of the whole architecture's behaviour, while the architecture is operating at runtime. The runtime inference is performed based on the Pearl's Message-Passing Algorithm [80] [81] [77].

4.3. Complexity Analysis

Given that the Bayesian analysis should be executed at runtime, this requires considering the complexity of the stability model.

The specification of conditional probability distribution $\mathsf{P}(\mathsf{X}_v|\mathsf{X}_{pa(v)})$ can be an intensive task, as the number of parameters grows exponentially with the size of $\mathsf{dom}(\mathsf{X}_{fa(v)})$. The complexity of the network is defined in terms of the family $fa(v)$ with the largest state space size $\|\mathsf{X}_{fa(v)}\| \triangleq |\mathsf{dom}(\mathsf{X}_{fa(v)})|$, where $fa(v) = pa(v) \cup \{v\}$. As the state space of a family of variables grows exponentially with the size of that family, a technique to reduce the complexity of Bayesian networks is to reduce the size of the parent sets $pa(v)$ to a minimum. This is, in fact, the case of the stability model, where the number of variables, i.e. stability attributes of each viewpoint, is limited. In such cases, estimating parameters from data could be a useful technique to simplify the intensive task of knowledge acquisition when operating at runtime.

While the Bayesian network is placed into operation, the model stores probability distributions and calculates various marginal distributions subject of

interest [82]. So, it is important to understand the storage capabilities of the network. Given that the variables are discrete and have a finite state space, to fully specify the model, we need to elicit $\mathsf{P}(\mathsf{X}_v)$ — which is the marginal probability mass function of X_v together with the conditional mass function $\mathsf{P}(\mathsf{X}_v|\mathsf{X}_{pa(v)})$ — of each of the variables conditioned on each possible configuration of values of its parents that might occur. The practical difficulty appears when the number of different configurations of parents, and hence the number of probability vectors that need to be elicited, is extremely large. In the case of a Bayesian network for a stability viewpoint, there is one variable subject of stability X_1, and its dependant variables $\{\mathsf{X}_2, \mathsf{X}_3, ..., \mathsf{X}_n\}$. If the number of possible stability values of X_1 is m_1 and for X_i is m_i, $i = 2, 3, ..., n+1$, then the number of probabilities we need to elicit $\mathsf{P}(\mathsf{X}_1)$ is $m_1 - 1$. And to elicit all the conditional tables $\mathsf{P}(\mathsf{X}_i|\mathsf{X}_{pa(v_i)})$ we need $m_i - 1$ for each possible stability value. Summing these, we have the total number of probabilities that need to be elicited, as follows:

$$m_1 \Big\{ \sum_{i=2}^{n+1} (m_i - 1 + 1) \Big\} - 1 \tag{3}$$

which is practically feasible, due the structure of the Bayesian network. Also, storing stability values in ranges, rather than single values, is useful for reducing the complexity of the stability model. For instance, response time is to be considered as ranges of 1-5, 5-10 ms. instead of multiple single values.

Considering the complexity of runtime inference, though probabilistic inference is an NP-hard problem in general, the complexity is polynomial in the number of variables of the network when the Bayesian network a singly connected garph [81] [75]. This is valid in the case of stability models, where we have one stability attribute directly connected to its dependent attributes for each stability viewpoint.

5. Methodological Support for Runtime Behavioural Stability

Our method for addressing architectural stability consists of three subsequent main phases: (i) stability analysis, (ii) stability modelling, and (iii) stability runtime support. For each step, we identify the human-based efforts required for the qualitative analysis and potential automated tools to be used in the quantitative modelling. The approach is illustrated in Figure 2.

Phase 1: Stability Analysis. In this phase, the initial analysis of stability as an architectural property is conducted, to build the *stability qualitative model*. In more details, this phase shall include the following activities:

Step 1. *identify stability dimensions.* For the case of self-adaptive software, the two main stability dimensions are: the adaptation goal and adaptation property. Other dimensions could be considered for the domain-specific application.

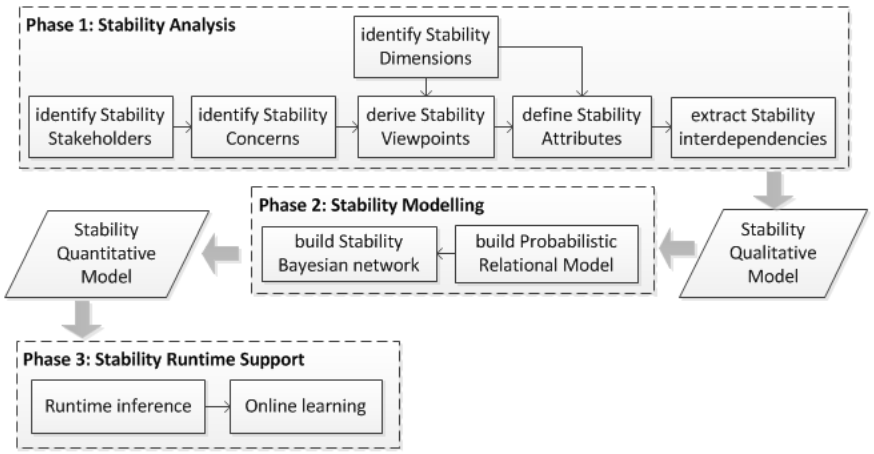


Figure 2: Architectural Stability methodology.

Step 2. *identify stability stakeholders.* Stability analysis entails architects to first identify the system's stakeholders that have interest in the system under consideration, and hence input for stability.

Step 3. *identify stability concerns.* In this step, the stability interests and concerns of stakeholders are taken into account in order to build a well-balanced solution, as it is important to have a good understanding of the different concerns that the stability analysis should reflect.

Step 4. *derive stability viewpoints.* Stakeholders concerns are consolidated to derive stability viewpoints, in order to consider stability from different perspectives for building a stability solution relative to multiple concerns. The analysis also takes into consideration concerns from the components of the self-adaptive system (i.e. the managed system and the adaptation controller).

Step 5. *define stability attributes and their evaluation criteria.* Stability attributes are, then, defined for different viewpoints reflecting the stakeholders' concerns for stability. The set of stability attributes also includes attributes belonging to the adaptation properties, as one of the main stability dimensions for self-adaptive software architectures. Evaluation criteria can inform the choice of suitable metrics for assessing the fulfilment of these attributes. The choice of the metrics is highly dependent on the analysis, where the metric can be structural, behavioural, quantitative, qualitative, economic-driven in nature. Practitioners often utilise commonly used metrics. ISO standards documents [36], guidelines and quality models [25] [83], white papers and benchmarks are among the credible sources for extracting these metrics. Systematic approaches could also be employed, such as goal-driven measurement [84] [85] and Goal Question Metric (GQM) approach [86].

Step 6. *extract interdependencies between stability attributes.* Interdependent quality attributes may influence one another. The dependencies between stability attributes are captured, in order to analyse how stabilising one attribute would affect the stability of related attributes.

The activities in this phase shall be conducted by architects and domain experts, relying on their experience and knowledge. The analysis could also be supported by subject matter expert panels and end-users workshops. The outcome of this phase is the *stability qualitative model* that will be used in the second phase as the model structure for quantitatively modelling stability.

Phase 2: Stability Modelling. In this phase, the stability model is built and stability attributes are quantitatively assessed. This phase includes the following activities:

Step 1. *build the probabilistic relational model.* For each viewpoint, a probabilistic relational model is built, based on the attributes dependencies identified in the last step of the stability analysis. Each probabilistic relational model, representing the relations between the attributes of a viewpoint, defines the structure of the Bayesian network. The problem of inducing the structure of Bayesian network is NP-complete, thus, heuristic methods are considered appropriate. Building the probabilistic relational models should go through an iterative process by the architects, domain experts and subject matter experts. This could be complemented with mechanisms and tool support to facilitate the adoption of the method. Examples include tools for documenting architecture knowledge and detecting patterns of use where similar problems could exhibit similar modelling, as well as platforms for sharing experiences, guidelines and recommended practices [87].

Step 2. *build the stability Bayesian network.* The Bayesian network is built for quantitatively modelling the interdependency impacts of different stability attributes. Bayesian network specifies the strengths of interdependencies and correlations between different attributes, using probability theory and preference relations quantified by the utility associated with these attributes. This task is inducing the Bayesian network for modelling stability by fusion of observed data and domain experts knowledge is undertaken. Building the Bayesian network could leverage on operational pre-experiments and/or simulations of the system.

Step 3. *build the stability Bayesian network.* The Bayesian network is built for quantitatively modelling the interdependency impacts of different stability attributes. Bayesian network specifies the strengths of interdependencies and correlations between different attributes, using probability theory and preference relations quantified by the utility associated with these attributes. This task is inducing the Bayesian network for modelling stability by fusion of observed data and domain experts knowledge is undertaken. Building the Bayesian network could leverage on operational pre-experiments and/or simulations of the system.

The outcome of this phase is the *stability quantitative model* that will be used for reasoning about stability during runtime. The model provides a basis for what-if analysis covering probable runtime behaviour that ranges from likely to extreme scenarios.

Phase 3: Stability Support at runtime. The stability quantitative model is used during runtime for estimating probable variations in stability attributes and associated trade-offs under the dynamically changing workload. This, consequently, improves the quality of decision making under runtime uncertainty for achieving the intended behaviour of the architecture and supports seamless runtime operation of the system. This phase is the continuous runtime process of:

Step 1. *conducting the runtime inference.* During runtime, posterior probabilities are obtained using the Bayesian network that allows measuring the probable effect of stabilising different stability attributes and their impact on each other. The posterior probabilities, contributing to the adaptation decision making, help in improving the quality of adaptation and ensuring the stability of quality attributes and hence the architecture intended behaviour.

Step 2. *performing online learning.* When the system is operating, new cases arise and it is recommended to learn from these cases, assuming that the structure of the Bayesian network will remain unchangeable [88] [75]. The conditional probabilities are dependent on the context and operation environment which change dynamically. The situation may also be that the simulation results used to extract the conditional probabilities do not reflect accurately the actual runtime workloads. This calls for online learning and updating the conditional probability distributions of the Bayesian network to reflect the real world, e.g. reasoning about quality requirements satisfaction as the system evolves dynamically [89] and learning for adaptation [90]. [2]

The runtime support for stability can be conducted online while the system is operating, by embedding the Bayesian analysis into the adaptation controller. The runtime inference and online learning can also be conducted through *symbiotic simulation* along with the adaptation controller. Symbiotic simulations shall run closely to the physical system, benefiting from real-time measurements from the actual system, and provide feedback to the system [91] [92] [93]. The results of the simulation shall be used for taking adaptation decisions autonomously during runtime by the adaptation controller (managing system). While the former approach can achieve effective immediate results, it can place extra computational overhead onto to the system while running. Conversely is the case of the latter approach. A balanced solution would be conducting

[2] An online learning algorithm would be out of the scope of this paper, but we introduce it in our method for the purpose of completeness. Our future work will focus on this part.

the inference online (using a threshold prior to violation) and employing the symbiotic simulation for online learning.

6. Case Study: Self-Adaptive Cloud Architectures

We show the applicability of the proposed approach through the case of self-adaptive cloud architectures built on the principles of self-awareness [94] [95]. First, we briefly introduce the architecture's domain, then apply the proposed approach for analysing and modelling runtime behavioural stability of this special class of self-adaptive architectures.

Cloud-based software architectures are a suitable example of high dynamism, unpredictability and uncertainty [96]. The execution environment of cloud architectures is highly dynamic, due to the on-demand nature of the cloud. Cloud architectures operate under continuous changing conditions, e.g. changes in workload (number/size of requests), end-user quality requirements, unexpected circumstances of execution (peak demand) [94] [14]. The on-demand service provision of clouds imposes performance unpredictability, and makes the elasticity of resources an operational requirement.

This type of architecture tends to highly leverage on adaptation (e.g. changing behaviour, reconfiguration, provisioning additional resources, redeployment) to regulate the satisfaction of end-user requirements under the changing contexts of execution [2] [14]. The self-adaptation process is meant to make the system behaviour converges towards the adaptation goals, i.e. quality requirements of the end-users [14]. An unstable adaptation will repeat the action with the risk of not improving or even degrading the system to unacceptable states [13]. Thus, there are more dynamics to observe, and stability is challenging with the continuous runtime adaptations in response to the perception of the execution environment and the system itself [14]. We consider the cloud architecture with a catalogue of architectural tactics, such as horizontal scaling (increasing/decreasing the number of physical machines), vertical scaling (increasing/decreasing the number of virtual machines or their CPU capacities), virtual machines consolidation (running the virtual machines on less number of physical machines for energy savings), as adaptation actions [95]. The purpose of adaptation is to satisfy the runtime demand of multi-tenant users, by changing configuration and choosing optimal tactics for adaptation.

Further, the economic model of clouds (pay-as-you-go) imposes on providers economical challenges for SLA profit maximisation by reducing their operational costs [96]. With the rising demand of energy, increasing use of IT systems and potentially negative effects on the environment, the environmental aspect, in terms of energy consumption, has emerged as a factor affecting the software quality and sustainability [97]. While sometimes imposed by laws and regulations, decreasing energy consumption does not have only potential financial savings, but also affects the ecological environment and the human welfare [97]. So, environmental requirements should be considered and traded off against business requirements and financial constraints [97].

The step-by-step application of the approach is described in the next subsections.

6.1. Stability Analysis for Self-adaptive Cloud Architectures

The analysis is based on the architectural stability taxonomy [72] applied on this case study [94] [95].

Step 1. *identify stability dimensions.* As mentioned in the Stability analysis model earlier, we identify two main stability dimensions for the case of self-adaptive software, which are: the adaptation goal and adaptation property.

Step 2. *identify stability stakeholders.* The main stakeholders that we consider are the end-users, the environment and the business.

Step 3. *identify concerns for stability.* The concerns for each stakeholder are listed as follows: (i) end-users' concern is the provision of QoS defined in their Service Level Agreements (SLAs), (ii) the environment regulations are concerned with the energy consumption constraints, and (iii) the business is concerned with operational costs.

Step 4. *derive stability viewpoints.* Given the stability dimensions and the stakeholders' concerns, we identify the following viewpoints for stability: quality of service, environmental, economical and quality of adaptation viewpoints. The former three denote the adaptation goals, and the latter represents the adaptation property dimension. The quality of service viewpoint mainly covers the quality requirements of end-users. The environmental viewpoint covers aspects related to energy consumption and savings [97] [98]. The economical viewpoint is related to the business concerns about monetary operational cost.

Step 5. *define stability attributes and their evaluation criteria.* Based on the stability viewpoints, we define related attributes. Stability attributes could, then, include traditional quality requirements specified in end-users SLAs. Here, we consider performance (measured by response time in milliseconds), and throughput (measured by number of completed requests per second). For the environmental aspect, we use the greenability attribute [97] [98] measured by energy consumption in kWh. For the economic constraints, we define the operational cost by the cost of computational resources (CPUs, memory, storage and bandwidth). Regarding the adaptation properties, we consider the settling time - that is the time required by the adaptation system to achieve the adaptation goal [13]. In order to capture the negative impact of adaptation on the system's behaviour, we consider the overhead of adaptation, measured by the frequency of adaptation cycles to achieve the adaptation goals [3] [13].

The analysis is illustrated in Figure 3. For simplicity, the information relating stability attributes with the stability concerns and their stakeholders is not included in the figure.

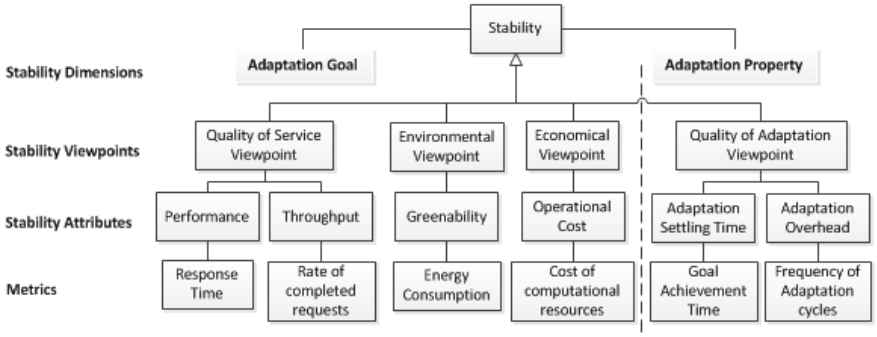


Figure 3: Case Study: Stability Analysis.

Step 6. *extract interdependencies between stability attributes.* Dependencies between stability attributes are defined based on the architect's domain experience, as depicted in Figure 4. For example, performance and greenability could contradictorily affect each other, i.e. stabilising performance shall demand more computational resources that consume more power and eventually have a negative effect on stabilising greenability. Meanwhile, greenability and operational cost could support each other, i.e. decreasing the usage of computational resources for saving power consumption would in turn decrease the operational costs.

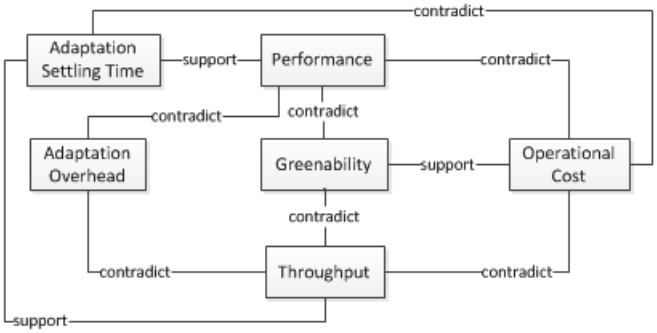
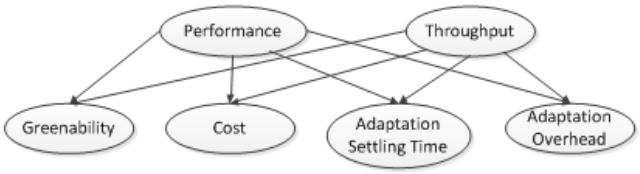


Figure 4: Case Study: Stability Attributes dependencies.

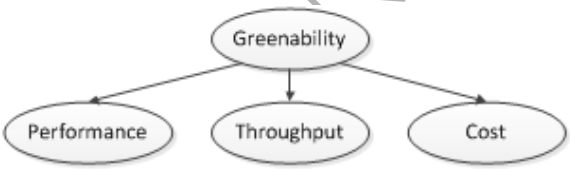
6.2. Stability Modelling for Cloud Architectures

Step 1. *build probabilistic relational model.* Based on the interdependencies between stability attributes, we deduce the probabilistic relational model

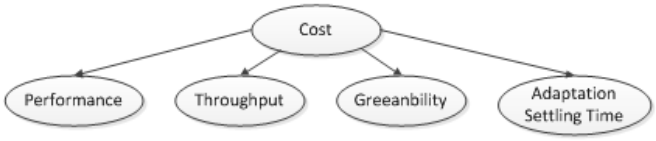
for each stability viewpoint. The probabilistic relational model related to the four stability viewpoints (quality of service, environmental, economical and quality of adaptation viewpoints) is depicted in Figure 5 (a), (b), (c) and (d) respectively. For instance, the quality of service model could be read as follows: stabilising the performance and throughput would affect the stability of related attributes that are greenability, operational cost and quality of adaptation attributes. Regarding the environmental model, such representation reflects that stabilising the energy consumption would affect the stability of performance, throughput and operational cost.



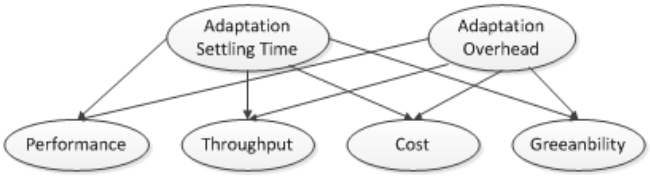
(a) Quality of Service viewpoint



(b) Environmental viewpoint



(c) Economical viewpoint



(d) Quality of Adaptation viewpoint

Figure 5: Case Study: Stability Relational Models.

Step 2. *build stability Bayesian network.* To quantitatively measure the depen-

dency factors between stability attributes and getting the prior knowledge to build the stability Bayesian network, we simulated the cloud self-adaptive architecture.

Our simulation tool, extending the widely adopted *CloudSim* simulation platform for cloud environments [99], was built using Java JDK 1.7 to create the cloud self-adaptive architecture, and was run on an Intel Core i5 3.40 GHz, 8 GB RAM computer. We conducted the pre-experiments by simulating a cloud data center initially operating 10 IBM x3550 server physical machines (PMs), with the configuration of 2 x Xeon X5675 3067 MHz, 6 cores and 16 GB RAM. The frequency of the servers' CPUs are mapped onto Million Instructions Per Second (MIPS) ratings: 3067 MIPS each core [100]. The characteristics of the virtual machines (VMs) types correspond to the latest generation of General Purpose Amazon EC2 Instances [101]. In particular, we use the m4.large (2 core CPU, 8 GB RAM) and m4.xlarge (4 core CPU, 16 GB RAM) instances. Initially, the VMs are allocated according to the resource requirements of the VM types. However, VMs utilise less resources according to the workload data during runtime, creating opportunities for dynamic consolidation.

For the energy consumption, we employed the power models defined in [100]. For a detailed cost model, the operational cost is calculated as a total of the usage of processing unit (0.04$ for CPU unit/sec.), memory (0.02$ for 1 GB memory/sec.), storage (1 GB storage/sec.) and bandwidth (0.01$ for 1 GB/sec.).

We run the pre-experiments for 300 time intervals, each interval is of 200 seconds, in order to get sufficient data for building the Bayesian network. In each time interval, we generate a random number of requests, and the length of each request randomly varies between 1,000 and 20,000 MIPS requiring 1 or 2 core CPU.

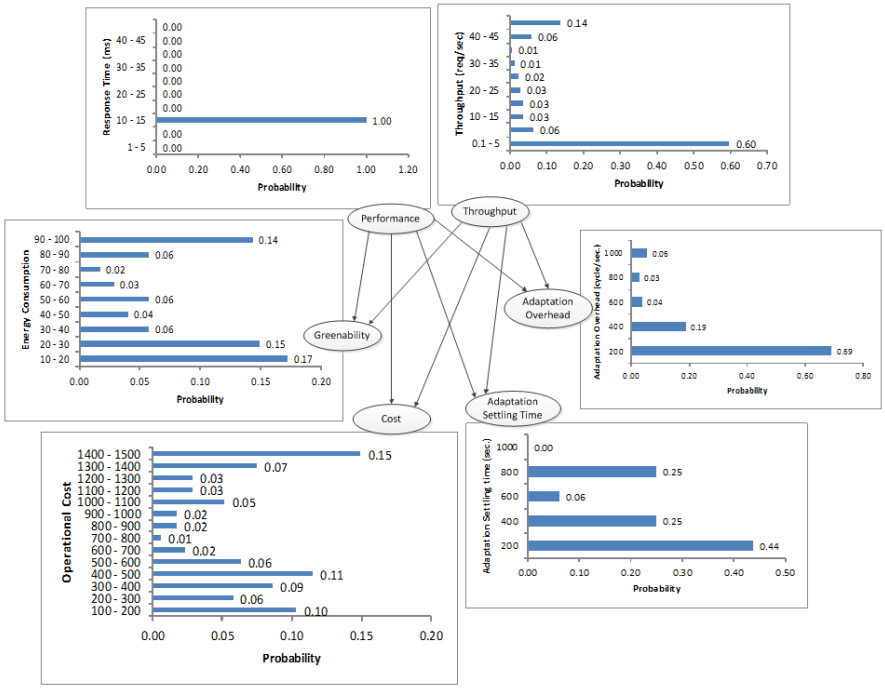
To measure the stability ranges for different viewpoints, we configured the architecture to take adaptation actions to stabilise specific attributes within different ranges, by setting this attribute as the single adaptation goal. We applied rule-based adaptations based on the stimulus-awareness level, i.e. the adaptation controller selects an adaptation tactic from the tactics catalogue mentioned earlier in order to achieve the quality requirement within the desired range whenever a violation is detected. The adaptation controller is responsible about selecting the adaptation tactic based on the quality attribute subject to violation, e.g. VM consolidation in case greenability is the stimulus. If multiple tactics could be applied, the rule applied is choosing the tactic that results in smaller overhead and/or cost [102]. For instance, we use vertical scaling first and then and horizontal scaling in the performance case, as the former is usually more expensive [102]. We, then, measured the impact of such stability actions on the stability of related attributes. Figure 6 shows the Bayesian networks of different viewpoints when stabilising their attributes for one range.

In more details, to capture stability from the quality of service viewpoint, we run the architecture with the adaptation goal of having the performance response time and throughput stable for different ranges. Figure 6 (a) shows the Bayesian network for the quality of service viewpoint when response time is stabilised for a range of 10-15 ms. and the throughput within a range of 1-5 request/sec. or higher. The impact of such stability actions is, then, measured on the related quality attributes, i.e. energy consumption, operational cost, adaptation settling time and overhead. The attributes selected for stability, i.e. performance, is indicated by probability = 1 for the range of 10-15 ms. Stabilising throughput was for the range 1-5 request/sec. in 60% of the cases or for higher ranges in the remaining cases, in order to reduce the complexity of adaptations during runtime. The probability distribution of impacts on related attributes is shown in the corresponding graphs of each node. As shown in the figure, this results in having energy consumption within ranges of 10-20, 20-30 and 50-100 kWh associated with probabilities 0.17, 0.15, 0.14 respectively, while other ranges have smaller probabilities. The cost ranges of 1400-1500, 400-500, and 100-200$ have the probabilities 0.15, 0.11, 0.10 respectively. Same applies for the adaptation settling time, where the highest probability is for the 200 sec., and the adaptation overhead where the majority of adaptation cycles took place every 200 sec.

With respect to the environmental viewpoint, Figure 6 (b) shows the probabilities of impacts of this viewpoint when the energy consumption is stabilised in the range of 50-60 kWh. The energy consumption range 50-60 kWh is indicated with probability = 1. As shown in the corresponding graph of each related attribute, the performance response time will range between 1-5 ms. with the highest probability of 0.49, the throughput range will be 0.1-5 request/sec. with a probability of 0.60, and the cost range 100-200$ with the higher probability of 0.25. The Bayesian network for economical viewpoint is shown in Figure 6 (c), where operational cost is stabilised for the range of 100-200$. Such stability would lead to having response time of 1-5 ms. with probability of 0.49, throughput of 1-5 request/sec. with probability of 0.60, energy consumption of 10-20 kWh with probability of 0.25, and adaptation settling time of 200 sec. with probability of 0.44. Other ranges for all attributes come with lower probabilities.

7. Experimental Evaluation

The main objective of the experimental evaluation is to examine the quality of service delivered and the quality of adaptation when employing the stability analysis during runtime, and assess the associated runtime overhead for the stability model. The experiments setup was inspired by the earlier work of Chen et al. [94] [102] on self-adaptive and self-aware cloud architectures.



(a) Quality of Service viewpoint

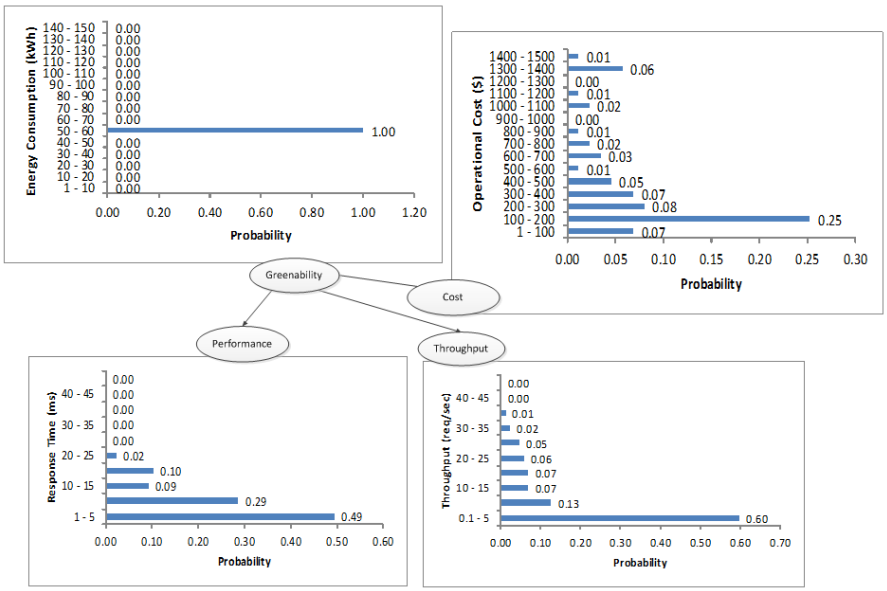
Figure 6: Case Study: Stability Bayesian networks (to be continued).

7.1. Experiments Setup

To simulate runtime dynamics, we used the *RUBiS* benchmark [15] and the *World Cup 1998* trend [16] in our experiments. The *RUBiS* benchmark [15] is an online auction application defining different services categorised in two workload patterns: the browsing pattern assuming read-only services, and the the bidding pattern simulating both read and write intensive services. For fitting the simulation parameters, we mapped the two service patterns of the *RUBiS* benchmark into MIPS, as follows: 10,000 MIPS 1 CPU core and 20,000 2 CPU cores for the browsing and bidding services respectively.

Instead of using random workload trend to simulate the number of requests, we varied the number of requests proportionally according to the *World Cup 1998* trend [16]. We compressed the trend in a way that the fluctuation of one day in the trend corresponds to 200 seconds in our experiments. Such benchmarks helped in stressing the architecture with highly frequent changing demand and observing its consequences on stability.

We simulated the cloud dynamics based on the *RUBiS* benchmark service patterns and run the entire *World Cup 1998* workload trend separately for each service pattern. For the experiment setup, the initial deployment for each service pattern of our experiments is shown in Table 1.



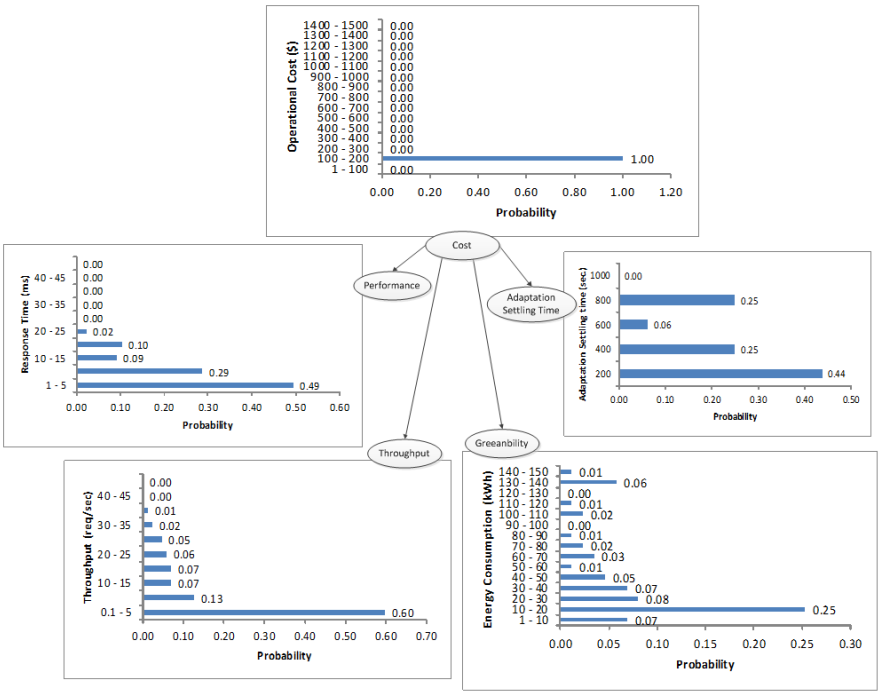
(b) Environmental viewpoint

Figure 6: Case Study: Stability Bayesian networks (to be continued).

Table 1: Initial deployments of the experimental evaluation

Configuration	**browsing service**	**bidding service**
No. of PMs	2	10
PMs type	IBM x3550 server	IBM x3550 server
PMs Specs	2 x Xeon X5675 3067 MHz, 6 cores, 16 GB RAM	2 x Xeon X5675 3067 MHz, 6 cores, 16 GB RAM
No. of VMs	6	30
VMs type	General Purpose Amazon EC2 Instances m4.large, m4.xlarge	General Purpose Amazon EC2 Instances m4.large, m4.xlarge
VMs Capacity	4 x 2 core CPU 8 GB RAM, 2 x 4 core CPU 16 GB RAM	20 x 2 core CPU 8 GB RAM, 10 x 4 core CPU 16 GB RAM

We performed the runtime stability analysis based on the following adaptations challenging goals: (i) response time not exceeding 15 ms., (ii) energy consumption not exceeding 50 kWh, and (iii) operational cost not exceeding 1000$. The runtime adaptation options are (by order of preference) as follows: increasing/decreasing the VMs capacity, increasing/decreasing the number of VMs, increasing/decreasing the number of PMs, and consolidating VMs on less number of PMs for shutting down the less busy PMs. The architecture was configured to select adaptations, informed by the stability analysis in one experiment and in another one without stability analysis, which we note as stability-based adaptations and conventional adaptations respectively. We, then, examined the quality of service provisioned and the quality of adaptation



(c) Economical viewpoint

Figure 6: Case Study: Stability Bayesian networks (cont.).

at each time interval of 200 sec. in both cases.

7.2. Examined Stability Attributes

The performance comparative results are summarised in Tables 2 and 3. Table 2 shows the violations of stability attributes of the adaptation goal. Performing adaptations informed by the stability analysis allowed to achieve less SLA violations for the attributes required to be maintained stable. For the browsing service case, stability-based adaptations succeeded to eliminate the violations for all stability attributes, while conventional adaptations still have violations for response time (0.28%), energy and cost (0.51% each). The case of bidding service, which requires higher computational requirements, succeeded to make less violations for response time (0.20%) and greenability (0.24%) compared with conventional adaptations (0.21% and 0.51% for both attributes respectively). Meanwhile, stability adaptations had 0.51% violations for energy consumption compared with 0.22% achieved by the conventional adaptations. This is interpreted by the less frequency of adaptations as clarified below.

Table 3 shows the quality of adaptation properties. Adaptation settling time in the case of stability-based adaptations is 300 and 200 sec. on average for the

Table 2: Violations of Stability Attributes Adaptation Goals (%)

Service Pattern	**Adaptation**	**Adaptation Goals**		
		Response Time	Energy Consumption	Operational Cost
browsing	stability-based	0.00	0.00	0.00
	conventional	0.28	0.51	0.51
bidding	stability-based	0.20	0.51	0.24
	conventional	0.21	0.22	0.51

browsing and bidding services respectively, compared with conventional adaptations (2040 and 2206 sec.). This reflects the capability of the stability analysis in converging the self-adaptive architecture quickly towards its adaptation goals, resulting in less quality of service violations. Regarding the adaptation overhead, stability-based adaptations succeeded to perform less frequent adaptations for both service patterns (3210 and 5800 adaptation cycles on average respectively), compared with the conventional adaptations (305 and 479 adaptation cycles on average). The less frequency of adaptation would leave the architecture in a more stable state and reflect the elimination of unnecessary adaptations. Meanwhile, this would require performing higher configurations for adaptations, in order to eliminate unnecessary frequent adaptations with less configurations. For instance, instead of increasing the number of PMs with 1 PM two times in 2 consecutive adaptations cycles, stability-based adaptations give insights for increasing 2 PMs in one longer adaptation cycle. This interprets the higher energy consumption appearing the case of bidding service above. The frequent adaptations shall result in slower settling time and consequently SLA violations. Generally, the less SLA violations, the quicker settling time, and the less frequent adaptations would reflect the behavioural stability state of the architecture.

Table 3: Stability Attributes Adaptation Properties

Service Pattern	**Adaptation**	**Adaptation Properties**	
		Settling time (sec.)	Overhead (frequency of adaptation cycles)
browsing	stability-based	300	3240
	conventional	2040	305
bidding	stability-based	200	5800
	conventional	2266	479

7.3. Complexity and Runtime Overhead

Considering the complexity and overhead of Bayesian analysis, the performance of the Bayesian network is directly related to the number of nodes [82]. In the case of stability model, the number of stability attributes of each viewpoint and their parents is limited. Thus, we can claim that running the stability model

during runtime is not an overhead on the system, compared with the expected benefits when achieving stability of the architecture.

With respect to the storage requirements of the Bayesian network, let us consider the case of the environmental viewpoint. There is one variable subject of stability X_1, i.e. greenability, and its dependant variables (performance, throughput, cost) $\{\mathsf{X}_2, ..., \mathsf{X}_{n+1}\}$, $n = 3$. If each attribute can take 10 possible ranges for stability, then we have 279 probabilities (from equation (3)), to be elicited in the stability model. For the case of the economical viewpoint, where cost is the variable subject of stability with 4 dependant variables, we have 359 probabilities to store in the stability network, which is just about practically feasible.

Meanwhile, the state space of the variables will grow exponentially, as the Bayesian network will accumulate knowledge during runtime. As such growth will affect the performance of the analysis during runtime, one possible technique could be prioritising the stability viewpoints. The highly-prioritised viewpoints could be kept running online, while the less prioritised could be considered offline using symbiotic simulations and adaptation decisions will be taken forward online. Such approach will limit the overhead of running the analysis at runtime, while preserve the benefits of the stability analysis.

7.4. Summary

Applying the stability approach allows closely monitoring for stability attributes, keeping their provision stable, as well as eliminating violations, consequent penalties and reputation loss. These attributes are the quality requirements critical for the system execution, such as response time for real-time systems. Adding economical requirements to the analysis allows the cloud providers to control their operational costs and, thus, maximise their revenues. Considering the quality of adaptation allows the system to converge towards adaptation goals, and hence eliminate SLA violations. Considering the overhead of adaptation in the stability analysis helps in eliminating unnecessary frequent adaptations that can lead to unstable states for the architecture.

Reaching stability state for the architecture given the runtime uncertainty would require determining an optimal adaptation action that achieves the stability of multiple attributes and converges quickly towards adaptation goals, while verifying the expected stability of interdependent attributes. Probabilistic Relational Models for different stability viewpoints have provided a natural representation for capturing the semantics of dependencies between different attributes subject of stability. Whereas, the Bayesian networks have presented a quantification of the dependence relations strengths and the preferences for runtime decision, as well as provided quantitative evaluation for reasoning under uncertainty.

Capturing dependency factors that affect the attributes subject to stability, the Bayesian networks for stability viewpoints provide a powerful decision-support tool, as they can be used to measure and predict the effect of an adaptation action on stabilising a specific attribute on other interdependent attributes. Given the knowledge gained from the model, this allows giving better insights

for runtime adaptations that would achieve architectural stability, where multiple attributes subject of stability are carefully considered. This also prevents unnecessary adaptations that could lead to instability.

Conducting stability inference for self-adaptive architectures, as part of their runtime operation, ensures more effective and efficient adaptations that contribute to the continuous fulfilment of quality requirements and eliminate SLA violations. As the objective of self-adaptivity is to seamlessly manage the runtime quality requirements and their trade-offs, the stability model allows verifying to what extent the adaptation actions are able to converge towards their goals, i.e. the quality of adaptation. Combining the adaptation properties with the adaptation goals in the process would result in a more efficient self-adaptive system. Compared with adaptations not informed by stability analysis, even with a multi-objectives optimisation, stability analysis would ensure the constant provision of these requirements with less violations, while the former might result in frequent unnecessary adaptations leading to instability.

Probabilistic modelling for multiple stability viewpoints allows reasoning about a stability state for the architecture that satisfies multiple attributes essential for stability. Such consideration would prevent SLA violations, excessive runtime adaptations, and consequently architecture drifting or phasing-out. The analysis can also give insights on possible enhancements of SLA parameters, as the analysis allows predicting how such enhancements will affect the stability of other attributes.

8. Discussion

The proposed analysis model is generic enough to be applied to architecture-centric software systems. As an example, the components of the self-adaptive software (i.e. managed system and adaptation controller) - appearing in the analysis model - could be replaced by domain-specific components of the architecture under evaluation. Yet, we demonstrated the proposed analysis model using self-adaptive software systems, as they tend to be more complex in behavioural aspects at runtime.

Domain-specific characteristics could be also included in the analysis as stability concerns and stability attributes, such as latency access for cloud federations [103]. As a general case, the stability analysis model could include any set of viewpoints and attributes subject of stability. Other stakeholders and stability dimensions could also be identified depending on the context, domain of the system and the architecture type.

On a wider perspective, the stability analysis approach could be applied to non-adaptive architectures for offline maintenance purposes. Architects can also employ the analysis model for making architectural decisions during design-time. Realising stability, as an architectural property, allows the architecture to preserve its capability to continuously meet its expected behaviour without phasing out.

Integrating the stability model into the adaptation process of such dynamic architectures provides valuable support for reasoning about the adaptation de-

cision and the operation of the architecture during runtime. The reasoning aims to satisfy not only the adaptation goals, but also ensure the constant provision beside the adaptation properties of the controller. The optimal of adaptations required tend to fulfil multiple stability properties and converge quickly towards adaptation goals. Such optimality of adaptation decisions can lead to the desirable stable state. However, it is possible to reach and maintain stability by reaching sub-optimal stages. Henceforth, the problem would be what is the range that can keep the system stable, which could vary between minimum sub-optimal to optimal. Yet, the results are sensitive to the analysis step and accuracy of data used to build the model.

Our method for reasoning about stability can make use of "sensitivity analysis" [104] [75], in order to test the extent to which small perturbations to the inputs of the model, i.e. entries of the conditional probability distributions, can affect the stability of the whole architecture. Two types of sensitivity analysis could be performed in probabilistic models: (i) evidence sensitivity analysis, in which how the result of of an evidence is sensitive to the variations in the set of evidences, and (ii) parameter sensitivity analysis, in which how the result of of an evidence is sensitive to the variations in a parameter of the model. Sensitivity analysis could be easily embedded in the steps of building the stability model.

Regarding the proposed methodological support, the level of automation generally varies between steps. For instance, the qualitative analysis depends on the human capabilities (stakeholders' input and architects' decision), which is different from the automated reasoning during runtime. Though extensive effort has been taken to ensure re-reproducibility of the method by providing systematic guidance, this would be subject to further empirical studies to determine the practicality of the method, where factors, such as availability of information, stakeholders' experience would be examined. Though, we believe that the presented case study for evaluation exemplifies the working procedure of the approach and reflect the potential usability.

The self-aware architecture —as the latest emerging class of self-adaptive architectures— could benefit from the different levels of awareness for realising stability. For instance, adaptation actions could be evaluated for stabilising the behaviour of the architecture towards multiple goals and interaction with other self-aware nodes. Meanwhile, the construction of the Bayesian network and the calculation of posterior stability probabilities during runtime could further benefit from the time-awareness level, which is the capability of having knowledge of historical and likely future phenomena.

9. Threats to Validity

Though we proposed a systematic approach for analysing and modelling stability, there are potential threats to validity:

- The dependency on the human capabilities in the analysis step of the proposed method would form a threat to validity on the end results when

using the proposed approach. This might be due to the lack of information or expertise knowledge. Yet, our approach could be complemented with formal methods of causality discovery [105] [106], structuring causal trees [80] and learning structure from data [81].

- With respect to the generalisability of the proposed work, we believe the method provides systematic guidance to architects and practitioners, where the steps were designed to promote replication. Yet, customisation might be needed if the self-adaptive system has different components for the adaptation controller. As the application of the method tends to be subject of the system under consideration, applicability and generalisability of the method to different software domains can uncover new modalities, customisation, simplification or extension to the method.

- The fact that the proposed method is evaluated by its authors presents a threat to objectivity. To mitigate this risk, we sought to conduct practical evaluation by architects in industrial settings, in order to provide more feedback from independent sources.

- Another threat to validity of our evaluation lies in the fact that the approach was evaluated using one case. Yet, the dynamics presented in cloud architectures is an appropriate case study representing dynamics of modern software systems, and we plan to conduct other case studies in industrial contexts and different business segments.

- Subjectivity might be considered a threat to validity for the stability analysis of the case study, as the analysis was conducted based on the authors' background and knowledge. Our strategy mitigation for this issue has been basing the case study on previous work of [94] [95] [72], this makes us believe that the case study is practical and reflects the nature of cloud-based software systems.

- Experiments were conducted in a controlled environment and have not considered the real-life scenario of switching between different service patterns and changing stability goals during runtime for different end-users. Given the use of a real-world workload trend and the *RUBiS* benchmark, we consider that our experiments have given good enough indication and approximation of likely scenarios in a practical setting. Also, we have chosen the stability goals thresholds purely based on our observations, e.g. response time not exceeding 15 ms. Yet, these goals have proved to be challenging when running the simulation for building the stability Bayesian networks.

10. Conclusion and Future Work

In this paper, we presented a systematic approach for analysing and modelling stability as an architectural property. The stability analysis, based on

architectural concerns and viewpoints, introduced a qualitative model for representing the knowledge related to the attributes subject to stability and their dependencies. For modelling stability, we employed probabilistic relational models that capture the correlations between stability attributes of different viewpoints. Bayesian networks are, then, used for quantitatively calculating probability distributions of the impact of stabilising specific attributes on interdependent attributes, as well as reasoning about stability under runtime uncertainty.

Our future work will focus on building the time-awareness model for online learning. More specifically, we will use historical information from the time-awareness level to update the prior knowledge and devise online learning for obtaining posterior stability probabilities. We will also consider modelling temporal (dynamic) relationships among stability attributes, i.e. representing how the value of an attribute may be related to its value and the values of other attributes at previous points in time.

Acknowledgment

Thanks are due to Patricia Lago for the constructive feedback to improve the technical quality and readability of the paper.

References

[1] B. Chen, X. Peng, Y. Liu, S. Song, J. Zheng, W. Zhao, Architecture-based behavioral adaptation with generated alternatives and relaxed constraints, IEEE Transactions on Services Computing (99).

[2] M. Salehie, L. Tahvildari, Self-adaptive software: Landscape and research challenges, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 4 (2) (2009) 1–42.

[3] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, J. Whittle, Software engineering for self-adaptive systems: A research roadmap, Springer-Verlag, 2009, pp. 1–26.

[4] R. de Lemos, H. Giese, H. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. Goschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezze, C. Prehofer, W. Schafer, R. Schlichting, D. Smith, P. Sousa, L. Tahvildari, K. Wong, J. Wuttke, Software engineering for self-adaptive systems: A second research

roadmap, Vol. 7475 of Lecture Notes in Computer Science, Springer-Verlag, 2013, pp. 1–32.

[5] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, A. L. Wolf, An architecture-based approach to self-adaptive software, IEEE Intelligent Systems 14 (3) (1999) 54–62.

[6] D. Garlan, Software architecture: a roadmap, in: Proceedings of Conference on The Future of Software Engineering, 2000, pp. 91–101.

[7] D. Garlan, Software architecture: a travelogue, in: Proceedings of International Conference on Future of Software Engineering, ACM, 2015, pp. 29–39.

[8] J. Buckley, T. Mens, M. Zenger, A. Rashid, G. Kniesel, Towards a taxonomy of software change, Journal of Software Maintenance and Evolution: Research and Practice 17 (5) (2005) 309–332.

[9] M. Salama, R. Bahsoon, P. Lago, Architectural stability: Survey of the state-of-the-art and research directions, (manuscript submitted for publication).

[10] R. Bahsoon, W. Emmerich, Evaluating software architectures: development, stability, and evolution, in: Proceedings of ACS/IEEE International Conference on Computer Systems and Applications, 2003, p. 47.

[11] R. Bahsoon, W. Emmerich, Architectural stability, Vol. 5872 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, book section 43, pp. 304–315.

[12] J. Kramer, J. Magee, Self-managed systems: An architectural challenge, in: Proceedings of Future of Software Engineering (FOSE), 2007, pp. 259–268.

[13] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, R. Casallas, A framework for evaluating quality-driven self-adaptive software systems, in: Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), ACM, 2011, pp. 80–89.

[14] N. M. Villegas, G. Tamura, H. A. Müller, Architecting software systems for runtime self-adaptation: Concepts, models, and challenges, Elsevier (Morgan Kaufmann), Boston, 2017, pp. 17–43.

[15] Rice University Bidding System (RUBiS).
URL `www.rubis.ow2.org`

[16] M. Arlitt, T. Jin, A workload characterization study of the 1998 World Cup web site, IEEE Network 14 (3) (2000) 30–37.

[17] International Organization for Standardization and International Electrotechnical Commission (ISO/IEC), ISO/IEC/IEEE 24765:2010(E) Systems and software engineering – Vocabulary, Report (2010).

[18] D. E. Perry, A. L. Wolf, Foundations for the study of software architecture, SIGSOFT Software Engineering Notes 17 (4) (1992) 40–52.

[19] M. Shaw, D. Garlan, Software Architecture: Perspectives on an emerging discipline, Prentice-Hall, Inc., 1996.

[20] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 2nd Edition, Addison-Wesley Longman Publishing Co., Inc., 2003.

[21] N. Medvidovic, R. N. Taylor, A classification and comparison framework for software architecture description languages, IEEE Transactions on Software Engineering 26 (1) (2000) 70–93.

[22] C. Seo, G. Edwards, S. Malek, N. Medvidovic, A framework for estimating the impact of a distributed software system's architectural style on its energy consumption, in: Proceedings of 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2008, pp. 277–280.

[23] A. Avizienis, J. C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing 1 (1) (2004) 11–33.

[24] V. T. Rajlich, K. H. Bennett, A staged model for the software life cycle, Computer 33 (7) (2000) 66–71.

[25] Software Engineering Standards Committee of the IEEE Computer Society, IEEE standard for a software quality metrics methodology, Report IEEE Std 1061-1998, The Institute of Electrical and Electronics Engineers, Inc. (1998).

[26] B. I. Witt, Software Architecture and Design : Principles, models, and methods, New York : Van Nostrand Reinhold, New York, 1994.

[27] H. Gomaa, Software Modeling and Design : UML, use cases, architecture, and patterns, Cambridge University Press, Cambridge, 2010.

[28] I. Sommerville, Software Engineering, Boston, Mass. London : Pearson Education, Boston, Mass. London, 2011.

[29] C. Lianping, M. A. Babar, B. Nuseibeh, Characterizing architecturally significant requirements, IEEE Software 30 (2) (2013) 38–45.

[30] P. R. Anish, B. Balasubramaniam, A knowledge-assisted framework to bridge functional and architecturally significant requirements, in: Proceedings of 4th International Workshop on Twin Peaks of Requirements and Architecture, ACM, 2014, pp. 14–17.

[31] R. Kazman, L. Bass, Toward deriving software architectures from quality attributes, Technical Report CMU/SEI-94-TR-010, Software Engineering Institute, Carnegie Mellon University (1994).

[32] D. Ameller, C. Ayala, J. Cabot, X. Franch, Non-functional requirements in architectural decision making, IEEE Software 30 (2) (2013) 61–67.

[33] K. J. Åström, B. Wittenmark, Adaptive Control, Addison-Wesley, 1989.

[34] R. Laddaga, Self-adaptive software, Technical Report 98-12, DARPA BAA (1997).

[35] A. C. Meng, On evaluating self-adaptive software, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 65–74.

[36] International Organization for Standardization and International Electrotechnical Commission (ISO/IEC), ISO/IEC 9126-1 – Information technology – Software product quality – Quality model, Report ISO/IEC 9126-1:2001, ISO/IEC (2000).

[37] L. Dobrica, E. Niemela, A survey on software architecture analysis methods, IEEE Transactions on Software Engineering 28 (7) (2002) 638–653.

[38] S. S. Yau, J. S. Collofello, Some stability measures for software maintenance, IEEE Transactions on Software Engineering SE-6 (6) (1980) 545–552.

[39] D. Bush, A. Finkelsteiin, Requirements stability assessment using scenarios, in: Proceedings of 11th IEEE International Conference on Requirements Engineering (RE), IEEE Computer Society, 2003, p. 23.

[40] S. S. Yau, J. S. Collofello, Design stability measures for software maintenance, IEEE Transactions on Software Engineering SE-11 (9) (1985) 849–856.

[41] M. E. Fayad, A. Altman, An introduction to software stability, Communications of the ACM 44 (9) (2001) 95–98.

[42] D. Grosser, H. A. Sahraoui, P. Valtchev, An analogy-based approach for predicting design stability of Java classes, in: Proceedings of 9th International Software Metrics Symposium, 2003, pp. 252–262.

[43] M. O. Elish, D. Rine, Indicators of structural stability of object-oriented designs: A case study, in: Proceedings of 29th Annual IEEE/NASA Software Engineering Workshop, 2005, pp. 183–192.

[44] D. Kelly, A study of design characteristics in evolving software using stability as a criterion, IEEE Transactions on Software Engineering 32 (5) (2006) 315–329.

[45] M. Jazayeri, On architectural stability and evolution, in: Proceedings of 7th Ada-Europe International Conference on Reliable Software Technologies, Springer-Verlag, 2002, pp. 13–23.

[46] S. A. Tonu, A. Ashkan, L. Tahvildari, Evaluating architectural stability using a metric-based approach, in: Proceedings of 10th European Conference on Software Maintenance and Reengineering (CSMR), 2006.

[47] A. Molesini, A. Garcia, C. v. F. G. Chavez, T. V. Batista, Stability assessment of aspect-oriented software architectures: A quantitative study, Journal of Systems and Software 83 (5) (2010) 711–722.

[48] A. Chomchumpol, T. Senivongse, Stability measurement model for service-oriented systems, in: Proceedings of 9th Malaysian Software Engineering Conference (MySEC), 2015, pp. 54–59.

[49] A. M. Lyapunov, The general problem of the stability of motion, Taylor & Francis, London, 1992.

[50] J. R. Leigh, Control Theory, 2nd Edition, Vol. 64 of IEE Control Engineering Series, Institution of Electrical Engineers, London, UK, 2004.

[51] T. Patikirikorala, A. Colman, J. Han, L. Wang, A systematic survey on the design of self-adaptive software systems using control engineering approaches, in: Proceedings of 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012, pp. 33–42.

[52] A. Gorbenko, V. Kharchenko, O. Tarasyuk, Y. Chen, A. Romanovsky, The threat of uncertainty in service-oriented architecture, in: Proceedings of RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems, ACM, 2008, pp. 49–54.

[53] R. Kazman, G. Abowd, L. Bass, P. Clements, Scenario-based analysis of software architecture, IEEE Software 13 (6) (1996) 47–55.

[54] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, in: Proceedings of 4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), 1998, pp. 68–78.

[55] T. Keuler, D. Muthig, T. Uchida, Efficient quality impact analyses for iterative architecture construction, in: Proceedings of 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2008, pp. 19–28.

[56] M. Neil, M. Tailor, D. Marquez, N. E. Fenton, P. Hearty, Modelling dependable systems using hybrid bayesian networks, Reliability Engineering & System Safety 93 (7) (2008) 933–939.

[57] D. Marquez, M. Neil, N. E. Fenton, A new bayesian network approach to reliability modelling, in: Proceedings of 5th International Mathematical Methods in Reliability Conference (MMR), 2007.

[58] A. Bobbio, D. Codetta-Raiteri, S. Montani, L. Portinale, Reliability analysis of systems with dynamic dependencies, John Wiley & Sons, Ltd, 2008, pp. 225–238.

[59] R. Roshandel, N. Medvidovic, L. Golubchik, A bayesian model for predicting reliability of software systems at the architectural level, in: Proceedings of Quality of Software Architectures 3rd International Conference on Software Architectures, Components, and Applications (QoSA), Springer-Verlag, 2007, pp. 108–126.

[60] J. Cámara, G. A. Moreno, D. Garlan, Stochastic game analysis and latency awareness for proactive self-adaptation, in: Proceedings of 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), ACM, 2014, pp. 155–164.

[61] J. A. Stankovic, H. Tian, T. Abdelzaher, M. Marley, T. Gang, S. Sang, L. Chenyang, Feedback control scheduling in distributed real-time systems, in: Proceedings of 22nd IEEE Real-Time Systems Symposium (RTSS), 2001, pp. 59–70.

[62] T. Patikirikorala, A. Colman, J. Han, L. Wang, A multi-model framework to implement self-managing control systems for QoS management, in: Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), ACM, 2011, pp. 218–227.

[63] J. L. Hellerstein, S. Singhal, Q. Wang, Research challenges in control engineering of computing systems, IEEE Transactions on Network and Service Management 6 (4) (2009) 206–211.

[64] International Organization for Standardization and International Electrotechnical Commission (ISO/IEC), ISO/IEC/IEEE 42010 – Systems and software engineering – Architecture description, Report ISO/IEC/IEEE 42010:2011(E), ISO/IEC (2011).

[65] B. Tekinerdogan, H. Sozer, Variability viewpoint for introducing variability in software architecture viewpoints, in: Proceedings of WICSA/ECSA Companion Volume, ACM, 2012, pp. 163–166.

[66] H. Koning, H. van Vliet, A method for defining IEEE Std 1471 viewpoints, Journal of Systems and Software 79 (1) (2006) 120–131.

[67] G. Qing, P. Lago, On service-oriented architectural concerns and viewpoints, in: Proceedings of Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA), 2009, pp. 289–292.

[68] P. Kruchten, R. Capilla, J. C. Dueas, The decision view's role in software architecture practice, IEEE Software 26 (2) (2009) 36–42.

[69] I. Ozkaya, R. Kazman, M. Klein, Quality-attribute-based economic valuation of architectural patterns, Technical Report CMU/SEI-2007-TR-003, Software Engineering Institute, Carnegie Mellon University (2007).

[70] P. Narman, M. Buschle, J. Konig, P. Johnson, Hybrid probabilistic relational models for system quality analysis, in: Proceedings of 14th IEEE International Enterprise Distributed Object Computing Conference (EDOC), 2010, pp. 57–66.

[71] A. J. Ramirez, A. C. Jensen, B. H. C. Cheng, A taxonomy of uncertainty for dynamically adaptive systems, in: Proceedings of 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE Press, 2012, pp. 99–108.

[72] M. Salama, R. Bahsoon, A taxonomy for architectural stability, in: Proceedings of 31st ACM/SIGAPP Symposium on Applied Computing (SAC), Software Architecture: Theory, Technology, and Applications Track (SATTA), 2016.

[73] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009.

[74] C. P. Robert, The Bayesian choice: from decision-theoretic foundations to computational implementation, Springer Science & Business Media, 2007.

[75] U. B. Kjaerulff, A. L. Madsen, Bayesian Networks and Influence Diagrams: A guide to construction and analysis, Information Science and Statistics, Springer, New York London, 2008.

[76] N. Bencomo, A. Belaggoun, V. Issarny, Bayesian artificial intelligence for tackling uncertainty in self-adaptive systems: The case of dynamic decision networks, in: Proceedings of 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013, pp. 7–13.

[77] R. E. Neapolitan, Learning Bayesian Networks, Pearson Prentice Hall, 2004.

[78] J. Pearl, Graphical models, causality, and intervention, Statistical Science 8 (3) (1993) 266–273.

[79] F. V. Jensen, Bayesian Networks and Decision Graphs, Springer-Verlag New York, Inc., 2001.

[80] J. Pearl, Fusion, propagation, and structuring in belief networks, Artificial Intelligence 29 (3) (1986) 241–288.

[81] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of plausible inference, Morgan Kaufmann Publishers Inc., 1988.

[82] J. Q. Smith, Bayesian Decision Analysis: Principles and Practice, Cambridge University Press, 2010.

[83] T. Hall, N. E. Fenton, Implementing effective software metrics programs, IEEE Software 14 (2) (1997) 55–65.

[84] R. E. Park, W. B. Goethert, W. A. Florac, Goal-driven software measurement. a guidebook, Report Technical Report CMU/SEI-96-HB-002, Software Engineering Institute, Carnegie Mellon University (1996).

[85] N. E. Fenton, S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Co., 1998.

[86] R. van Solingen, V. Basili, G. Caldiera, H. D. Rombach, Goal Question Metric (GQM) Approach, John Wiley & Sons, Inc., 2002.

[87] P. Lago, P. Avgeriou, P. Kruchten, Fifth international workshop on sharing and reusing architectural knowledge (shark), in: Proceedings of ACM/IEEE 32nd International Conference on Software Engineering (ICSE), Vol. 2, 2010, pp. 437–438. doi:10.1145/1810295.1810417.

[88] F. V. Jensen, An Introduction to Bayesian Networks, UCL Press, London, 1996.

[89] C. Ghezzi, G. Tamburrelli, Reasoning on non-functional requirements for integrated services, in: Proceedings of 17th IEEE International Requirements Engineering Conference (RE), 2009, pp. 69–78.

[90] M. Hölzl, T. Gabor, Reasoning and Learning for Awareness and Adaptation, Springer International Publishing, 2015, pp. 249–290.

[91] H. Aydt, S. J. Turner, W. Cai, M. Y. H. Low, Research issues in symbiotic simulation, in: Proceedings of Winter Simulation Conference (WSC), 2009, pp. 1213–1222.

[92] S. J. Turner, Symbiotic simulation and its application to complex adaptive systems (keynote), in: Proceedings of IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2011, pp. 3–3.

[93] B. Tjahjono, J. Xu, Linking symbiotic simulation to enterprise systems: Framework and applications, in: Proceedings of 2015 Winter Simulation Conference (WSC), 2015, pp. 823–834.

[94] T. Chen, F. Faniyi, R. Bahsoon, P. R. Lewis, X. Yao, L. Minku, L. Esterle, The handbook of engineering self-aware and self-expressive systems, Technical report, School of Computer Science, University of Birmingham (2014).

[95] M. Salama, R. Bahsoon, Quality-driven architectural patterns for self-aware cloud-based software, in: Proceedings of IEEE 8th International Conference on Cloud Computing (IEEE CLOUD), 2015, pp. 844–851.

[96] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of Cloud Computing, ACM Communications 53 (4) (2010) 50–58.

[97] P. Lago, S. A. Kocak, I. Crnkovic, B. Penzenstadler, Framing sustainability as a property of software quality, Communications of the ACM 58 (10) (2015) 70–78.

[98] C. Calero, M. Piattini, Introduction to green in software engineering, Springer, 2015, pp. 3–27.

[99] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[100] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurrency and Computation: Practice and Experience 24 (13) (2012) 1397–1420.

[101] Amazon Web Services, Inc., Amazon EC2 Instance Types, accessed: 2016-06-01.
URL `https://aws.amazon.com/ec2/instance-types/`

[102] T. Chen, R. Bahsoon, Symbiotic and sensitivity-aware architecture for globally-optimal benefit in self-adaptive cloud, in: 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), ACM, 2014, pp. 85–94.

[103] A. N. Toosi, R. N. Calheiros, R. Buyya, Interconnected cloud computing environments: Challenges, taxonomy, and survey, ACM Computing Surveys 47 (1) (2014) 1–47.

[104] A. Ekrt, S. Z. Nmeth, Stability analysis of tree structured decision functions, European Journal of Operational Research 160 (3) (2005) 676–695.

[105] P. Spirtes, C. Glymour, R. Scheines, Causation, prediction, and search, 2nd Edition, MIT Press, Cambridge, Mass., 2000.

[106] S. Shimizu, P. O. Hoyer, A. Hyvärinen, A. Kerminen, A linear non-gaussian acyclic model for causal discovery, Journal of Machine Learning Research 7 (2006) 2003–2030.