Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap

Conference Paper · June 2016

DOI: 10.1109/SCC.2016.113

2 authors:

Sara Hassan
Birmingham City University
6 PUBLICATIONS **200** CITATIONS
SEE PROFILE

Rami Bahsoon
University of Birmingham
223 PUBLICATIONS **3,455** CITATIONS
SEE PROFILE

Microservices and Their Design Trade-offs: A Self-Adaptive Roadmap

Sara Hassan
School of Computer Science
University of Birmingham
Birmingham, UK
ssh195@cs.bham.ac.uk

Rami Bahsoon
School of Computer Science
University of Birmingham
Birmingham, UK
r.bahsoon@cs.bham.ac.uk

***Abstract*—Migrating to microservices (*microservitization*) enables optimising the autonomy, replaceability, decentralised governance and traceability of software architectures. Despite the hype for microservitization , the state of the art still lacks consensus on the definition of microservices, their properties and their modelling techniques. This paper summarises views of microservices from informal literature to reflect on the foundational context of this paradigm shift. A strong foundational context can advance our understanding of microservitization and help guide software architects in addressing its design problems. One such design problem is finalising the optimal level of granularity of a microservice architecture. Related design trade-offs include: balancing the size and number of microservices in an architecture and balancing the non-functional requirement satisfaction levels of the individual microservices as well as their satisfaction for the overall system. We propose how self-adaptivity can assist in addressing these design trade-offs and discuss some of the challenges such a self-adaptive solution. We use a hypothetical online movie streaming system to motivate these design trade-offs. A solution roadmap is presented in terms of the phases of a feedback control loop.**

***Keywords*-microservices; trade-offs; non-functional requirements; granularity; self-adaptativity; decision-making**

I. Introduction

Several industries have recently migrated to or consider migrating to microservices [1] (microservitization). Servitization is a shift towards utility-based engineering for software products into services that are continuously motivated by adding long-term value to the system users [2]. We therefore view microservitization as a form of servitization where services/components are transform into microservices — a more fine-grained and autonomic form of services — to add long-term value to the architecture. Microservitization is also an example of a paradigm shift since it involves a dramatic change to the way software is designed and developed within a firm [3]. Microservitization is applicable to "brownfield" (i.e., existing systems migrating to microservices) and "greenfield" (i.e., building new systems) development [1].

Microservitization involves isolating business functionalities into microservices that interact through standardised interfaces. Isolating business functionalities aims at optimising the autonomy and replaceability of the service(s). It can also facilitate autonomous management (i.e., decentralised governance) of the service(s). Therefore the paradigm shift can promote better traceability, accountability and auditing for the service(s) and their provision in the event of failure. These qualities are examples of potential value to be added to the software architecture through microservitization by enhancing its flexibility to cope with operation, maintenance and evolution uncertainties. Ultimately, this can also relate to improved maintenance costs and cost-effective quality of service (QoS) provision to system users.

Despite the hype and the business push towards microservitization [1], there is a lack of academic consensus regarding the definition and properties of the paradigm shift and corresponding design patterns for microservices [4]. One of the motivations of this paper is to bridge this gap by digesting the various informal views from industry on defining microservices to aid in understanding the design problems faced by software architects when migrating to microservices.

Among these problems is ***finalising the level of granularity of a microservice*** too early. "Splitting too soon can make things very difficult to reason about. It will likely happen that you (the software architect) will learn in the process. [1]." This problem is of significance both in brownfield and greenfield development [1] since it affects the choice of concrete realisations (and thereby microservice vendors) when instantiating a system's abstract architecture.

II. Motivating Example and Contribution Definition

We use the context of greenfield development here as an example to motivate two aspects (or trade-offs) of the granularity problem. We use a hypothetical online movie streaming subscription-based system for illustration. The Promise requirements data repository [5] was used to derive the functional and non-functional requirements of this system. This repository provides indicative requirements from different application domains. We will use the following indicative functional requirements (FRs):

- ***FR1***: System shall allow users to view reviews of selected movies by other users.
- ***FR2***: System shall allow users to add their own movie review for a selected movie.

- ***FR3***: System shall allow the administrator to approve a review posted by a user.

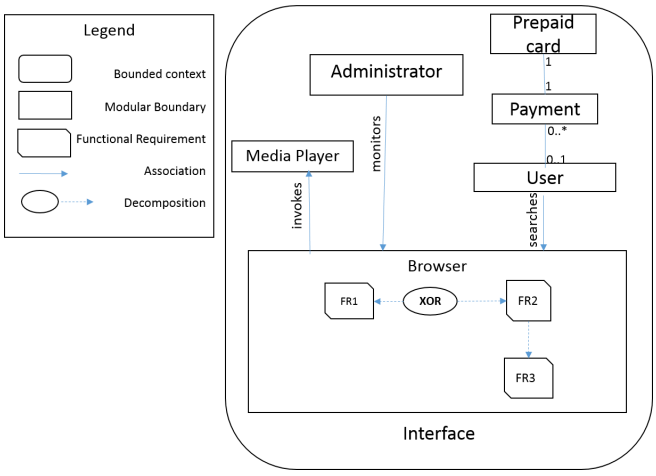

Figure 1. Segment of the abstract architecture for online movie streaming subscription-based system provided by the architect

At design time, we assume the initial abstract architecture of the system is as shown in Figure 1. Each modular boundary ultimately maps to a single microservice.The questions that arise when encapsulating the three FRs above within modular boundaries are:

1) Would too much communication complexity be introduced if movie reviews are encapsulated within a separate modular boundary?
2) Is the functionality isolation enhanced by separating reviews into a separate modular boundary worth the investment?
3) How does encapsulating the reviews affect the non-functional requirement (NFR) satisfaction of the overall system (global NFR)?
4) Would encapsulating the reviews into a new modular boundary affect the NFR satisfaction of the individual (local) modular boundaries interacting with it (namely *Browser*, *User* and *Administrator*)?

The questions above present two trade-offs that need to be considered when finalising the level of granularity of a microservice.

The former two questions represent a trade-off which we refer to as ***the size versus number of microservices*** [4] trade-off. Intuitively, the more microservices introduced into the architecture, the higher the level of isolation between the business functionalities. This comes at the price of increased network link communications and perhaps increased object distribution complexity. Addressing this trade-off systematically is essential for assessing the extent to which "splitting" is beneficial regarding the potential value of microservitization (as defined above).

The latter two questions represent another trade-off which we refer to as ***the local versus global NFR satisfaction*** trade-off. The third question is concerned with the provision of QoS to system users, which is ultimately a reflection of global NFR satisfaction. That is, the microservitization exercise can not be done without keeping human and beneficiaries of the services in the loop. For example, a user of the movie streaming system will only be concerned with the results of his/her request returning within 5 seconds of issuing it. This requirement is a global NFR for the system. He will not be concerned with the graphics function within an implementation of the *Browser* modular boundary returning within 1 second of its execution. This latter requirement is a local NFR. It is these local NFRs that interact together to satisfy the global NFRs of the system however. This interaction takes different forms depending on the number of microservices and the pattern in which they interact. Addressing this trade-off systematically means more cost-effective QoS provision to the users of the system on the long term.

The optimal balancing point for both trade-offs highly depends on the current scenario in which the system is operating (i.e., its current environment) [6]. For example, for a steadily large volume of requests for reviews on a particular movie, directing those requests to a separate *Review* modular boundary is reasonable to avoid the *Browser* becoming a bottleneck for the whole system. This level of granularity can be perceived as the optimal one for the current environment. On the other hand, if there is a persistent outage in a network link connecting the *Browser* and the *User*, then merging the 2 modular boundaries can help reduce the latency submitting a request for a movie review. In this scenario, a coarser rather than finer level of granularity can be perceived as the optimal one. Therefore, the argument here is that ***aggressive isolation of business functionalities is not necessarily ideal for all scenarios of the environment***.

Furthermore, the definition of the optimal balancing point depends on the relative NFR preferences as elicited from the system's stakeholders and the relative criticality of the local and global NFRs. For example, a choosing the coarse level of granularity in the example above is only deemed the optimal balancing point if the system stakeholders actually consider reducing the latency of the request (a global NFR) critical compared to the cost of implementing this coarse grained architecture. In particular this cost can be negative implications on local NFRs (e.g., the return time of a specific function within a microservice will increase). Therefore, only if the global NFR of reducing request latency is critical to the stakeholders compared to other (possibly conflicting) global and local NFRs will this coarse level of granularity be deemed optimal.

The link between the local/global NFR satisfaction trade-off and the microservice size/number trade-off can be inferred from the discussion above. We argue that determining the optimal level of granularity for a microservice architec-

ture entails addressing the microservice size/number trade-off. Deciding between functionally equivalent architectures each having different numbers and sizes of microservices (i.e.,different levels of granularity of microservices) requires knowledge of the implications of each candidate architecture on NFR satisfaction. In particular, it requires knowledge of how adjusting the size of an individual microservice affects its NFR satisfaction and how that adjustment of size affects the NFR satisfaction of the overall system.

The contributions of this paper are thereby three-fold:

- We review and reflect on the definition, properties, and modelling techniques of microservices as presented in informal literature (Section III).
- In Section IV, we formulate the problem of addressing the design trade-offs and introduce our solution proposal.
- A self-adaptive solution is then outlined at a high level in Section V. The movie streaming system is used as a running example to illustrate the applicability of existing techniques in the literature for each phase. The envisioned challenges and possible future research directions towards the proposed solution are then discussed in Section VI.

III. Reflection on the State of the Art and Practice in Microservices

Due to the recency of the research area, there is a multitude of ways in which microservices have been defined and modelled. Clear understanding of the paradigm shift, its motivations, and implications are prerequisites for advancing microservitization. We have reviewed the state of art and practice to capture the different ways in which microservices can be defined and modelled.

In [7], microservices are defined as an architectural style which promotes developing an application "as a suite of small services, each running in its own process and communicating with lightweight mechanisms [7]". In [8], this definition is slightly complemented to focus on the "weight" of a microservice rather than its size. In particular, a microservice has to be lightweight enough to be replaceable, rather than having to invest in maintaining it. A more compact definition of the microservices is "small, autonomous services that work together [9, p.2]." Autonomy is the ability "to change independently of each other, and be deployed by themselves without requiring consumers to change [9, p.3]." A healthy implication of autonomy is decentralised governance of the system [7]. Teams are fully responsible for a specific microservice(s), making responsibility for design decisions more traceable. The responsibility spans designing, building, testing, deploying and providing support for that service(s) [1]. They have full choice of the technology used to implement a microservice(s), as long as the interface facing the user is standardised.

We therefore view microservices as ***autonomic, repleaceable and deployable artefacts of microservitization that encapsulate fine-grained business functionalities presented to system users through standardised interfaces. The autonomy of these artefacts allows for governing them in a decentralised manner and tracing their changes.***

It is suggested in [1], [8] to model microservices in an event-driven manner. An "event" is formulated as the delta that has occurred in the service due to a change in the environment. An event bus captures events occurring in upstream services, then the downstream services can capture them later. This allows each downstream service to only capture events that it is interested in from a stream of events [4], allowing for autonomy and decentralised governance. Furthermore, the implementation of the event bus is independent from the services capturing events from it, allowing for replaceability. Microservices can also be reasoned about in terms of domain models. In [10], a domain model is made up of bounded contexts, each encapsulating a subset of business functionalities. This subset is further broken into more fine-grained modular boundaries. It is these modular boundaries which are candidates to be mapped to individual microservices. The internals of bounded contexts and modular boundaries do not need to be exposed to the rest of the system, allowing for autonomy, replaceability and decentralised governance.

The literature reveals only a subtle distinction between the service-oriented architectural style (SOAs) and microservices. However, this distinction needs to be made more explicit to highlight the uniqueness of microservices. The uniqueness comes from: 1) the potential of microservices as autonomous fine-grained computational units and, 2) the enhanced flexibility of their underlying style. Therefore, microservices can be thought of as "enriched"and more modularised services. Classic dynamic service selection and composition in SOAs mostly focuses on dynamic restructuring of services while ensuring the overall functionality of the system. However, existing approaches do not question the granularity problem — whether the service provision is at the optimal level of granularity or not. We aim to exploit that distinction by addressing the microservice size/number trade-off and the local/global NFR satisfaction trade-off systematically.

IV. Problem Formulation

Among the crucial and non-trivial decision problems (DPs) that constitute addressing the size/number and the local/global NFRs satisfaction trade-offs are the following:

1) ***DP1:*** A solution which manages these trade-offs has to determine (given the current environment scenario and the stakeholders' definition of "optimality" regarding the trade-offs of concern):
   - When does ***decomposing*** a microservice into more fine-grained ones achieve the required optimality for both trade-offs?

- When does ***merging*** several fine-grained microservices into a coarse-grained one achieve the required optimality for both trade-offs?
- When should the current level of granularity be kept ***without further merging or decomposition***?

2) ***DP2:*** The chosen architecture still has to ***guarantee the functional requirements*** of the system, regardless the level of granularity of the microservices in that architecture.
3) ***DP3:*** Much of the ***uncertainties*** that relate to the choice of the optimal architecture and the knowledge that relates to the expected behaviour of the system [11], [12] can not be fully captured at design time.

DP2 motivates developing a solution that addresses the trade-offs at a more abstract service composition level rather than that followed by classical contributions on service composition. This abstraction allows for focussing on the encapsulation of functionalities rather than the particular concrete services that will instantiate these functionalities.Abstract services represent units of computation and their composition through well-defined interfaces [13]. An abstract service can be operationalised by several alternative concrete services . Referring to Figure 2, consider a situation where the software architect produces an abstract specification of a service-oriented system, S. For S, there are $S'_1 \ldots S'_n$ refined abstract architectures that vary in the number and size of their microservices. Also, each S'_i varies in the way it balances between the global and local NFR satisfaction. It is this refined abstract architecture solution space (i.e., $S'_1 \ldots S'_n$) that we aim to manage and reason about to choose the optimal S'_i given a current environment scenario and NFR preferences. Given the choice of S'_i, this refined abstract architecture can then be mapped at runtime to concrete services from a service registry or marketplace (i.e., a classic service selection venture starts given the output of the solution we are proposing).

DP1 and ***DP3*** call for runtime support to continuously update design-time knowledge. This requires monitoring the environment surrounding the system and using the monitored data to update the system's knowledge to better cater for uncertainties. This support can be provided by engineering self-adaptivity [12] as a solution. Therefore, we argue that inducing a microservices architecture with the primitives of self-adaptivity is a strong candidate for addressing these trade-offs. Many of the concerns that drive the choice of the optimal architecture are of runtime nature and are merely difficult to anticipate and analyse at design time. This self-adaptive solution however inevitably takes as input the architect's best knowledge at design time of what the optimal level of granularity of the architecture would be. The role of the self-adaptive solution is thereafter to support, refine and update this knowledge at runtime. A self-adaptive system is made up of an adaptation system which is responsible for planning any adaptations when need be, and a managed system which executes these plans [14].

There are several mechanisms which the adaptation system can adopt [12], [14]. Underlying most of these mechanisms is the concept of control loops (MAPE-K loop) [12], [14]. We propose a solution based on the MAPE-K loop to render a systematic solution that improves over the system lifetime through accumulating knowledge.

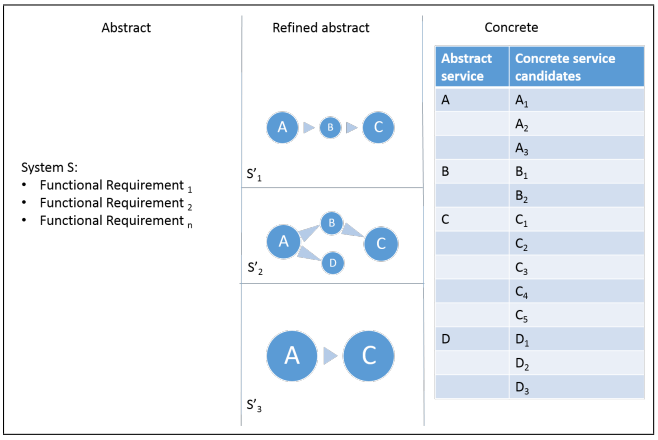

Figure 2. The different conceptual modelling levels for microservices

V. Solution Roadmap

This section proposes and discusses the suitability of techniques in the literature to each phase of the MAPE-K loop.

1) **Input to MAPE-K loop**: Architectural modelling techniques [15] provide several ways to capture an architect's best knowledge of the optimal abstract architecture S (referring to Figure 2) with different levels of expressiveness. This vision will correspond to the managed system of the self-adaptive solution we are proposing. Iterations of capturing and elicitation are envisioned here due to the decentralised governance culture of the microservices setting.
2) **Monitoring**: Defining variables to monitor at runtime and relating them to NFRs is a two-fold process: 1) an NFR gathering and elicitation phase and 2) linking the NFRs to variables to be monitored as runtime. Furthermore, there are 2 types of variables that need be monitored: variables that reflect the scenario (e.g., client request rate) and variables that reflect the system's behaviour in that scenario (e.g., response time). We envision utility trees (such as those presented in [16]) as a possible technique for both phases due to their understandability.
   We appreciate however that they lack the exhaustiveness of other more powerful techniques such as [17] where utility trees are leveraged with implied scenario detection.
3) **Analysis**: Several analysis techniques are presented in [11] to interpret the values of the monitored variables into a

perception of the current scenario surrounding the managed system. The enhanced distribution of functionalities introduced by microservitization poses a challenge to this phase. This distribution increases the possibility of wrong perceptions of the current environment scenario across the microservices, therefore increasing the likelihood of triggering faulty adaptation decisions.

4) **Planning**: Given the interpretation of monitored data from the analysis phase, the decision to trigger (and plan) an adaptation is taken in this phase. Deciding to trigger an adaptation to begin with depends on the relative NFR preferences captured from stakeholders.
   Once a decision is made to trigger an adaptation, the solution space to address ***DP1*** is a set of functionally equivalent refined abstract architectures of microservices. The level of autonomy of the self-adaptive solution that we are proposing depends on who is responsible for the formulation of this solution space. If the solution space is provided as input to the self-adaptive solution then the role of the self-adaptive solution is less autonomous and more of a design support for decision making. In particular, the planning phase would then only be responsible for choosing the optimal architecture given the solution space from the software architects. On the other hand, a more autonomous self-adaptive solution is one where it builds the solution space of refined abstract architectures at runtime and chooses the optimal one from that space. The field of artificial intelligence (AI) planning provides techniques to support an autonomous planning phase through dynamic, recursive service decompositions techniques governed by a set of predicates [6]. The first step in [6] is translation of a composite web service specification to the problem domain, therefore allowing abstracting away from concrete specifications to more abstract technology independent ones (thereby addressing ***DP2***). Manual synthesis of the solution space governed by workflow patterns [18] is a possible direction in case a design support for decision making rather than an autonomous self-adaptive solution is adopted.
   Once the solution space is formed, software repositories can be mined or economic models can be consulted can be consulted to predict the NFR satisfaction levels for each refined abstract architecture and the added value that each can bring to the system in terms of the properties referred to in Section I, coming to a decision regarding the optimal refined abstract architecture (S'_i).
5) **Execution**: Classic concrete service selection techniques can be employed in this phase to map the refined abstract architecture to concrete services. Crucially however, the concrete service selection process can be potentially simplified by pruning the solution space. For example, if the chosen S'_i suggests only 3 microservices, then only 3 service markets need to be examined instead of 5 or 6 markets. In addition, this phase needs to feedback into the knowledge base of the adaptation system. This knowledge update will feed into more informed decision making over the lifetime of the system (addressing ***DP3***). The key challenge to this phase is the availability of specialised service markets to instantiate all of the microservices in the chosen refined abstract architecture.
6) **Knowledge**: Probabilistic modelling is an attractive venue to capture dynamic uncertainty in the beliefs about the system at runtime. Bayesian probabilities is particularly attractive since it captures the delta in probability given some condition, so it can be used to capture updates in knowledge objectively. The challenge here however is the effort required to infer the prior and posterior probabilities of NFR satisfaction needed to capture the delta in knowledge.

VI. Discussion and Future Work

Although Section IV poses interesting research problems, we appreciate that the trade-offs presented in Section I can be indicators of other design problems in addition to finalising granularity. For example, lack of balance in the local/global NFR satisfaction trade-off or a significant increase in complexity (as mentioned in ***DP1***) might be indicators of sub-optimal deployment choices (e.g., choice of communication protocols and data access dependencies). Therefore, one of our future work directions is to refine our understanding of the causal relationship between the granularity problem and its indicators. We are aiming to do that by conducting a series of experiments a concrete system adopting microservitization, in each experiment altering either a deployment choice only or the granularity of the microservices only and thereafter monitoring the effect of these alterations on different QoS variables.

Once this relationship is clarified, the longer term research direction is providing a systematic self-adaptive solution inspired by the proposal in Section IV. We appreciate that the microservitization paradigm shift poses research challenges to each phase of the MAPE-K loop. For example, defining and prioritising the variables to be monitored at runtime is a significant challenge in the granularity problem due to the extra effort required to prioritise NFRs and variables in both the local and global architectural levels (as discussed in Section I).

Motivated by these challenges and others, an initial step towards a systematic solution is deriving the concrete challenges to each phase of the MAPE-K based on experimentation with a runnable system (with more concrete FRs) that adopts microservitization. In particular, we aim to outline a repeatable solution roadmap in our future work as we derive the challenges of each phase from a concrete example system. Once these challenges are reified, we will assess the suitability of examined techniques in the literature to addressing these challenges in the microservices setting

through experimentation and monitoring with the concrete example system.

We also appreciate that integrating a self-adaptive, runtime solution into a running system raises practicality issues. Therefore we envision potential in leveraging on symbiotic simulation approaches [19] to implement the MAPE-K loop. Data from the monitoring phase is fed into a simulation of the managed system, where the analysis and planning are carried out to assess the reliability of the chosen refined abstract architecture before it is embedded in the running system.

VII. Conclusion

Our contribution has reflected on the informal literature about microservices, their properties and their modelling techniques. We have then formulated the problem of finding the optimal level of granularity during microservitization, motivating the problem using a a hypothetical online movie streaming subscription-based system. We break down the problem into 2 trade-offs: the microservice size/number and the global/local NFR satisfaction. A self-adaptive runtime solution to the problem is then proposed and discussed. Some of the research challenges and practicality issues that microservitization poses to such a solution are highlighted along with possible future research directions.

References

[1] Z.Dehghani, "Zhamak dehghani real world microservices: Lessons from the frontline," https://youtu.be/hsoovFbpAoE, Youtube, feb 2015.

[2] A. B.School, "The aston centre for servitization research and practice," http://www.aston.ac.uk/aston-business-school/research/centres/aston-centre-for-servitization-research-and-practice/.

[3] B. H. C.Cheng and J. M.Atlee, "Research directions in requirements engineering," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 285–303. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.17

[4] F.George, "Challenges in implementing microservices by fred george," aug 2015.

[5] "The promise repository of empirical software engineering data," http://openscience.us/repo. North Carolina State University, Department of Computer Science, 2015.

[6] M.Vukovic and P.Robinson, "Shop2 and tlplan for proactive service composition," in *UK-Russia Workshop on Proactive Computing*, feb 2005.

[7] M.Fowler and J.Lewis, "Microservices a definition of this new architectural term," http://martinfowler.com/articles/microservices.html, March 2014.

[8] Y.Sader, "A microservices reference architecture," https://www.youtube.com/watch?v=KHqMPRA6jVI, Youtube, 2015.

[9] S.Newman, *Building Microservices*, 1st ed. O'Reilly Media, feb 2015.

[10] E. J.Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*.

[11] M.Sama *et al.*, "Multi-layer faults in the architectures of mobile, context-aware adaptive applications: A position paper," in *Proceedings of the 1st International Workshop on Software Architectures and Mobility*, ser. SAM '08. New York, NY, USA: ACM, 2008, pp. 47–49. [Online]. Available: http://doi.acm.org/10.1145/1370888.1370901

[12] B.Cheng *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science, B.Cheng *et al.*, Eds. Springer Berlin Heidelberg, 2009, vol. 5525, pp. 1–26. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02161-9_1

[13] P.Dolog, M.Schfer, and W.Nejdl, "Design and management of web service transactions with forward recovery," in *Advanced Web Services*, A.Bouguettaya, Q. Z.Sheng, and F.Daniel, Eds. Springer New York, 2014, pp. 3–27.

[14] R.deLemos *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, ser. Lecture Notes in Computer Science, R.deLemos *et al.*, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 1–32. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35813-5_1

[15] M. A.Babar *et al.*, *Software Architecture Knowledge Management - Theory and Practice*, 1st ed. Springer-Verlag Berlin Heidelberg, 2009.

[16] R.Kazman, M.Klein, and P.Clements, "Atam: Method for architecture evaluation," http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5177, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2000-TR-004, 2000.

[17] F.Faniyi *et al.*, "Evaluating security properties of architectures in unpredictable environments: A case for cloud," in *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, June 2011, pp. 127–136.

[18] W. M. P.Van Der Aalst *et al.*, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, Jul. 2003.

[19] O.Onolaja, R.Bahsoon, and G.Theodoropoulos, *Trust Management V: 5th IFIP WG 11.11 International Conference, IFIPTM 2011, Copenhagen, Denmark, June 29 – July 1, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ch. Trust Dynamics: A Data-Driven Simulation Approach, pp. 323–334. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-22200-9_26