Software Self-adaptation and Industry: Blame MAPE-K

Rogério de Lemos
University of Kent, UK
r.delemos@kent.ac.uk

***Abstract*—If software self-adaptation has to be widely adopted by industry, we need to think big, embrace complexity, provide easily deployed and cost-effective solutions, and justify trust. On fairness, MAPE-K should not solely take the blame. MAPE-K is one of the many interpretations of feedback loops apply to systems for which mathematical models - mostly based on control theory, are difficult to be synthesised. MAPE-K has provided a basic and widely accepted framework for justifying the deployment of feedback loops in software systems. Undoubtedly, it has helped to promote and advance the whole area, but now more concrete and resilient solutions are necessary. This position paper argues that, first, industry has been adopting software self-adaptation, perhaps in a way that may not be recognised by the academic community, second, generic solutions are unfeasible since every software system brings its own challenges, and thirdly, the generic stages associated with a feedback loop, like MAPE-K, are insufficient to solve specific challenges.**

***Index Terms*—feedback loop, MAPE-K, micro-services, human-in-the-loop, complexity, modelling, resilience**

I. Introduction

Although there exists a wide range of reasons for industry to be slow in adopting processes, techniques and mechanisms emerging from the self-adaptive software systems community, in this position paper, we focus on three fundamental reasons, namely, industrial pragmatism, nature of software, and MAPE-K loop [1].

The incentives for embracing self-adaptation can be considered from two perspectives: development and operational. From the *development perspective*, part of the software evolution lifecycle is shifted into operational time, and this has implications on how software is developed, deployed and evolved. From the *operational perspective*, it is the ability of software to handle uncertainty at run-time. While the latter is usually considered as the main benefit of self-adaptation, the former is the price that needs to be paid to obtain that benefit since software needs to be built in a different way. The inertia for industry to be more receptive towards self-adaptation is associated with the fact that software development practices need to be changed in a cost effective way.

Before expanding the three identified reasons for industry to be slow in embracing self-adaptation, we provide below a brief introduction to these. The first reason is what we identify as *industrial pragmatism*, that is, if it is not good enough why bother to change? This is not restricted to the quality of the solutions, but also depends on the timing of the solutions. Integrating new technologies into an industry can often take time due to the additional costs involved, which can slow down the realisation of the benefits.

The second reason that we identify is the *nature of software*. Software can be incorporated or transformed into almost anything, and to try to come up with generic solutions for controlling a wide range of target software systems is a huge enterprise.

Finally, the third reason is the *MAPE-K loop*. The quest for implementing the four stages of the MAPE-K loop has hindered the development of processes, techniques and mechanisms for supporting self-adaptation. The end-to-end argument between probes and effectors of the MAPE-K loop has steered towards monolithic controllers, rather than flexible controllers that would be more appropriate for self-adaptive software systems. The motivation for this is that, software self-adaptation requires a wide range of specific tasks, and combination of these, that goes beyond the abstract four stages of the MAPE-K loop, which can be inspirational but not practical.

In the following, we delve in more detail into each of the above identified reasons.

II. Industrial Pragmatism

A key aspect of self-adaptive software systems is to champion the use of explicit feedback control loops for avoiding the intricacies and risks of implicit feedback loops [5]. However, software implementation of implicit feedback loops is quite natural since these can easily be integrated into the software systems. Abstracting away the controller from the targeting system is challenging since it is quite difficult to separate, at the code level, 'what' the software does from 'how' should it be done [1]. Implicit control loops, in addition to be more natural to code, they are cheaper and faster since they bundle up the subject and its master, with a total disregard to technical debt. On the other hand, explicit feedback control loops require further system designs in which the manipulation of models becomes a necessity, and this further complicates the case for explicit feedback loops.

The reality is that, industry takes a more pragmatic approach when developing software. Either they take the lead and come up with their own solutions, or they are extremely selective on what should be embraced. However, it is a mistake to put in a single bucket all types of industries, and their approaches to software development. There is a wide range of industries,

[1] Purposely keeping away for those identified by ChatGPT: complexity, cost, reliability, lack of standards and trust.

and each may face distinct goals and constraints, which usually shape their pragmatics. Since software is making inroads into a wide range of novel applications, those industries that rely on software-based autonomy could be the first ones to embrace self-adaptation based on explicit feedback loops.

The incorporation of a controller into a software system, to make it self-adaptable, is something that needs to be considered right from the software inception. It should be an integral part of the system design, and not an afterthought. Software can always be refactored [1], but this invariably is one off situation. For a sustainable development of self-adaptive software systems, there is the need for industry to embrace the development of software systems that are truly resilient.

In summary, it could be argued that industry has been quite cautious in adopting some of the technologies coming out from the engineering for self-adaptive software system community perhaps because in their perception, these technologies do not seem to bring significant benefits that justify industry to change their practices. However, this may be revised if there is a shift to move evolution costs to operational time that relies on self-adaptation.

III. Nature of Software

A key challenge when adapting software systems is that each software has their own peculiarities in terms of its structure, the services it provides, and the quality of these. Since software lacks a strong mathematical underpinning, as physical laws and rules of operation, it is difficult to synthesise mathematical models that would allow to develop generic solutions that could be easily manipulated. The search for a formal underpinning has been the holy grail, in terms of safety and security, when collecting formal evidence for building up assurance cases [4].

Faced with a problem domain in which every system has its own idiosyncrasies, we should not expect to be able to come up with generic solutions. If there is no space for generic solutions that can be adopted across different application domains or systems, it becomes difficult for industry to embrace a new technology that for many still has some drawbacks, mainly regarding assurances, when incorporating self-adaptation. Hence it should not be a surprise if industry comes up with tailored made solutions, depending on the application domain.

In summary, the nature of software hinders the search for effective generic solutions that would enable controlling software adaptation at run-time. Even if we had the ability of manipulating formal representations that would enable the synthesis of generic controllers, that would not be enough since there are other layers on which these controllers depend.

IV. (Not so) Inspirational MAPE-K Loop

Although MAPE-K loop, and all existing variants, have promoted the use of feedback control loops in software systems, they nevertheless trivialised the search for effective solutions. The reason for this is in the details that each of the MAPE-K stages abstracts from.

Some of the existing generic controllers, like Rainbow [3], have been effective in controlling some of aspects of software systems, which can either be structural or parametric. However, if the evolutionary software lifecycle is mapped into the four stages of the MAPE-K loop, new activities need to be associated with each of the stages [2]. For example, the analysis stage may include model checking or testing architectural configurations, These specific activities that go beyond the control of the target system, may need themselves to be self-adaptive. The monolithic nature of the MAPE-K loop has inspired current controllers, however these need to be replaced by flexible structures that can be reconfigured according to the needs of the target software system. For industry, it would be easier to handle smaller controllers that would enable flexible structures, rather than monolithic complex controllers that rely on technologies that are not so accessible.

In summary, although monolithic generic detectors will continue to be relevant for a certain class of problems, a promising solution, for example, would be to implement controllers as an ensemble of service-specific micro-controllers.

V. Conclusions

Whatever reasons there are for industry not to adopt the scientific and technological contributions emanating, mainly, from the self-adaptive software systems community, we shall not be dishearten. We should strive to continue to find new ideas, and push the boundaries of knowledge in order to enable, in a future not too far, the successful industrial deployment of self-adaptive software systems that are resilient. For this to be achieved, it is fundamental to engage with industry to understand what is really important for them. However, if after all the effort little progress is made, we shall wait, patiently, for sure our time will arrive.

As a concrete outcome of this exercise, of questioning why industry has been reluctant in embracing the contributions from the community of self-adaptive software systems, we should at least be able to identify other reasons beyond what is currently accepted, and come up with pathways for confronting challenges ahead.

References

[1] J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. R. Schmerl, and R. Ventura. Incorporating architecture-based self-adaptation into an adaptive industrial software system. *J. Syst. Softw.*, 122:507–523, 2016.

[2] R. de Lemos and P. Potena. Chapter 14 - identifying and handling uncertainties in the feedback control loop. In I. Mistrik, N. Ali, R. Kazman, J. Grundy, and B. Schmerl, editors, *Managing Trade-Offs in Adaptable Software Architectures*, pages 353–367. Morgan Kaufmann, Boston, 2017.

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

[4] P. J. Graydon, J. C. Knight, and E. A. Strunk. Assurance based development of critical systems. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 347–357, 2007.

[5] C. Perrow. *Normal accidents*. Basic Books, New York, NY, 1984.