The Unknown Unknowns Are Not Totally Unknown

David Garlan
Institute of Software Research, Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
garlan@cs.cmu.edu

Abstract**—** *The question of whether "handling unanticipated changes is the ultimate challenge for self-adaptation" is impossible to evaluate without looking closely at what "unanticipated" means. In this position paper I try to bring a little clarity to this issue by arguing that the common distinction between "known unknowns" and "unknown unknowns" is too crude: for most systems there are changes that are not directly handled by "first-order" adaptation, but can, with appropriate engineering, be addressed naturally through "second-order" adaptation. I explain what I mean by this and consider ways in which such systems might be engineered.*

Keywords—adaptive systems, uncertainty, unknowns

I. Introduction

Self-adaptive systems (SAS) are, by definition, designed to handle at run time specific changes in the system, the environment or the context of use. Typical examples of changes addressed by today's SAS include varying loads on the system, resource variability, system faults, intrusions, etc. However, as with any control system, a self-adaptive system is designed to adapt to certain kinds of changes for which it can provide stable outcomes; outside that envelope all bets are off. The former are often termed "anticipated"; the latter "unanticipated".

In characterizing this situation, it is tempting to place changes into two buckets, often referred to as the *known unknowns* and the *unknown unknowns*. The former are the anticipated ones, for which the adaptive system is designed to handle; the latter for which it provides no guarantees. We engineer a system to adapt to the known unknowns; we rule out the others as being outside the scope of our engineering effort.

We argue that this is too crude a categorization and that there is an interesting space of unanticipated changes and uncertainties that fall outside the envelope of self-adaptation for which a system was primarily designed, but at the same time are known well-enough that they can be identified and handled systematically. We describe an architectural approach to doing this and outline some of the engineering techniques that can be used to realize that approach.

II. Known Unknown Knowns

All control systems are designed to deal with certain environmental perturbations (and not others) in order to maintain some set point. Such scoping assumptions are often key to the achievability and efficiency of the control function they are to perform, and there is a considerable body of knowledge that allows us to design and quantify the properties of such control.

Self-adaptive systems, viewed as a kind of control system, are no different: we engineer them to handle certain uncertainties that are not known until run time. The ability to characterize and bound those uncertainties allows us to build efficient mechanisms for adaptation and to reason about them [2]. These are the known unknowns, or what might be termed "first-order" concerns.

But what about the changes that fall outside the scope of the engineered self-adaptive system – the unknown unknowns? Today's practice is largely to ignore them: we may not be able to detect them; even if we can detect them, our models may lack expressiveness to reason about how to deal with them ; and even if we can reason about remediation, we may not have appropriate mechanisms to actually carry out an effective adaptation.

But let's look closer. Rather than throwing up our hands, perhaps we should ask how we might engineer systems to deal with a subset of these unknown unknowns – what one might term the *known unknown unknowns*. The key idea that I'd like to suggest is that we can engineer second-order systems in which the adaptive capabilities of a system are themselves adapted to these known unknown unknowns. Thus we can envision a multi-layered system, consisting of the managed system, the managing layer, and a meta-managing layer.[1] The meta-management (or second-order) layer adapts the self-adaptive system based on meta-models of the (first-order) SAS and its environment.

To create a second-order adaptive system, however, one must engineer the first-order SAS to be itself adapted. Based on current research, and inspired by other systems that are robust to unexpected changes, here are some engineering strategies that could be used individually and in combination to achieve this.

1) **Over-engineer**: build in mechanisms to handle more than we expect. This is a common engineering practice for NASA space missions and bridge engineering, which build in considerable "head room" to handle situations that were not anticipated. In the context of SAS this might include things like additional sensors that are not normally needed, reserve resources that we can call on, adaptation tactics that continue to function even outside the range for which they were designed.

2) **Build meta-interfaces**: if a higher order system is to adapt a SAS, it needs to have an interface to monitor and affect

[1] We could consider higher layers, of course, but that's beyond the scope of this topic.

the SAS. Such an interface needs to be designed in, and appropriately abstract models are needed to support the meta layer. [1] for example, proposes the use of higher-order adaptation policies – essentially adaptation policies and strategies that are parameterized and can be changed at run time by a meta-manager.

3) **Assume less**: It is possible to build adaptation mechanisms that have a broad domain of applicability outside some particular range of changes by assuming there is more uncertainty in our knowledge than may be the case. As an example, autonomous cyber defenses are often designed to handle certain specific kinds of attack, but run aground when faced with exploits or attacker profiles that were not known when those defenses were built. It is possible, however, to build in approaches that make few assumptions about prior knowledge of the attacker and system vulnerabilities to reason more generally about graceful degradation under *any* attack [4].

4) **Create reusable parts**: It is possible to design a SAS from reusable building blocks that can be reassembled in new ways if the need arises. For example, recent work has investigated how adaptation strategies can be automatically carved up into reusable "chunks" that can be genetically recombined to produce new adaptation strategies to handle unforeseen changes [3].

III. Conclusion

The observant reader will notice that in this paper I haven't actually taken a stand on the question posed for debate. This is because I don't think it is actually worth debating: instead, I would argue that we should ask the question, how do we make our self-adaptive systems more robust to changes in behavior, environment, and context that are not the primary first-order management (or control) concern of the system. In this paper I have tried to sketch out a possible direction from an architectural point of view, and suggest engineering approaches to realize it.

Acknowledgment

This research was supported by the National Science Foundation (CCF-1618220), the NSA (H9823018D0008), and the Office of Naval Research (N00014172899).

References

[1] Thomas J. Glazier, David Garlan and Bradley Schmerl. Case Study of an Automated Approach to Managing Collections of Autonomic Systems. In *Proceedings of the 2020 IEEE Conference on Autonomic Computing and Self-organizing Systems*, August 2020.

[2] Javier Cámara, Alessandro V. Papadopoulos, Thomas Vogel, Danny Weyns, David Garlan, Shihong Huang and Kenji Tei. Towards Bridging the Gap between Control and Self-Adaptive System Properties. In *Proceedings of the 15th International Symposium on Software Engineering for Adaptive and Self-managing Systems*, June 2020.

[3] Cody Kinneer, Rijnard Van Tonder, David Garlan and Claire Le Goues. Building Reusable Repertoires for Stochastic Self-* Planners. In *Proceedings of the 2020 IEEE Conference on Autonomic Computing and Self-organizing Systems,* August 2020.

[4] [WGF21] Ryan Wagner, David Garlan and Matt Fredrikson. Architecture-based Graceful Degradation for Security. Unpublished.