Achieving an acceptable design model for autonomic systems

Simon Dobson
Systems Research Group
School of Computer Science and Informatics
UCD Dublin IE
simon.dobson@ucd.ie

Abstract

Autonomic systems present unique design challenges, in that their individual adaptive components may interact in complex ways which defeat traditional approaches to design, analysis and implementation. We argue for a more holistic approach to design, and identify some key properties that are necessary for next-generation design methods.

1 Introduction

Autonomic systems present unique opportunities and unique challenges. Their flexibility and adaptability must be balanced against the need for predictability and the ability to satisfy (and be seen to satisfy) the requirements of customers.

The engineering of adaptive systems is a formidable challenge – especially as many software engineers would argue that we have so far failed to conquer the problem of engineering systems to exhibit a single well-defined behaviour! Whilst great progress has been made in individual areas such as adaptive network protocols, we still have very little idea how such complex and adaptive components can best be used within larger systems.

The purpose of this paper is to explore what it is that makes autonomic systems design different from other, more traditional domains. In doing so we will derive some properties that a design model for autonomic systems should have. This is not to suggest that a single, unified model with universal applicability exists: rather the opposite. However, there are core issues which *any* model must address, and by identifying these issues we hope to achieve some clarity as to the requirements under which we must work.

Our central thesis is that, for modern systems design, *everything interesting is composition*: the interconnectedness of modern enterprise and communications systems means that we cannot consider individual applications or services in isolation, and this especially includes their adaptive behaviours. This implies that we must focus on the ways in which systems compose together and interact, and ensure that we achieve predictable and robust composition even in the face of dynamic populations of adaptive services.

Section 2 explores some of the challenges in autonomic systems design, and especially the sources of costs of ownership for complex systems. As a result, section 3 argues that we should treat interactions between components, rather than components themselves, as the object of study for design, and identifies some core areas on which to focus design effort. Section 4 concludes with some possible directions for the future.

2 Composition as the core challenge

Autonomic computing and communications arose in response to two related driving forces [6]. The complexity of systems is increasing, driven by increased use of automation, increased use of pervasive and sensor systems, and increased interconnectivity between systems previously regarded as independent. This in turn increased the total cost of construction and (more importantly) ownership for large systems. The cost of particular concern was the failure cost, since increased interconnection means that failures tend to propagate beyond their initial causes.

The use of the term "autonomic" comes by analogy with biological systems, in which the autonomic nervous system is responsible for regulating sub-conscious activities such as organ function in such a way as to allow conscious activity to proceed unimpeded. If we decide to run (a conscious decision), our breathing and heart rates will increase (unconsciously) to accommodate us; if we enter a hot room, unconscious mechanisms will begin to cool us.

The key concept in autonomic systems is the "autonomic control loop" (figure 1) which captures the feedback nature of adaptivity. The system collects data about its environment and functioning, which is the analysed and used to inform an adaptation decision. Once executed, the impact of

IEEE TFAAS Newsletter – Issue 7 (Oct/Nov 2006) – Letters

http://tab.computer.org/aas

the adaptation may be observed and used to inform further decisions. This closing of the decision loop characterises autonomic systems: decisions reflect the impact of previous decisions.

What does this analogy mean for computers? Much of the work in autonomic systems involves improving the automated management of technology. Examples include:

- improving the self-description of components to allow automated installation;
- improving network protocols to adapt to changes in other layers of the network stack [7]; and
- changing protocols at a fundamental level to provide more adaptive responses to changing network conditions [5].

Such approaches tackle local complexity: a specific challenge is identified, has its complexity reduced by a particular adaptive technique. There are significant bodies of research in applying autonomics-inspired techniques to computing and communications [3].

2.1 The costs of change

What guarantees do we have, however, that a collection of such approaches *collectively* reduces complexity? Using two of the examples above, does adding cross-layer information flow improve or damage protocols that are adaptive at the network layer? The actual answer – which may be yes or no, of course depending on the details of the individual approaches – is not important for our present purposes: the point is rather that we have no reliable way of deciding *a priori* one way or the other. This means that, to continue the example, we need to design our network system *en bloc* and determine (through analysis or testing) that it provides the properties we want. If we change one of the components, we must re-analyse and re-test.

The software engineer Bertrand Meyer once characterised changes in software as falling into one of two classes. *Linear* changes are those whose cost to perform is proportional to the size of the *change being made*: a large significant change is costly, while a small change is (relatively) cheap. *Non-linear* changes, by contrast, have a cost proportional to the size of the *system being changed*, so large systems may be prohibitively expensive to change in any way whatsoever.

The significance of this classification to autonomics is clear. Systems arise from, and are characterised by, the interactions between components rather than from the behaviours of those components individually. The adaptive behaviour of a single component may interfere with that of another, so that the composition of two individually-correct behaviours is no longer correct. This renders all compositions, and all changes, non-linear and hence fundamentally infeasible as systems grow.

At present, while we can adequately design adaptive individual adaptive components to address well-bounded problems, we have considerably less understanding of the ways in which these components compose. The interactions of composed components frequently come as a surprise: their interactions are likely to be sub-optimal at best. This does not provide at attractive basis for the design and evolution of autonomic systems.

2.2 A shift of focus

Software engineering is based on the principle of *decomposition*: given a complex problem, we decompose it into smaller problems, continuing recursively until we have sub-problems that are simple enough to be solved in isolation. However, the tacit assumption here is that the sub-problems will compose to solve the original problem we had.

We focus on generating simple, solvable problems from complex ones. *Is it possible that, for modern (and especially autonomic) systems, this focus is wrong?*

If we accept that component interactions are complex and unpredictable, this means that they should be given priority in the design process. This is especially true if we consider that the component populations of systems will change over their lifetimes, without (it is hoped) adversely affecting the system itself.

Software engineering teaches us to seek a correct decomposition of a problem into an orthogonal, weakly-coupled collection of cohesive components which are then implemented. The objects of study are the individual components, the things that solve the "real" sub-problems.

However, an autonomic system cannot realistically be treated as a simple collection of sub-problems. Any action we take in one area may impact on another, positively or negatively. If the components are adaptive themselves, then their adaptations may interact in different ways over time. It doesn't matter how effective a component is at addressing its own problem: if it damages the behaviour of another component, the system as a whole will behave sub-optimally.

The implication is clear: *the correct object of study for complex adaptive systems is not the components, but their interactions*. At the risk of taking an overly extreme position, it almost *doesn't matter* what an individual component does, and how effective it is at solving its local problem: the component will often be replaced, or will interact poorly with some other component either now or later when that component is itself replaced. The properties of the individual components – their efficiency, performance, elegance, robustness and so on – are not the issue: rather, *we need*

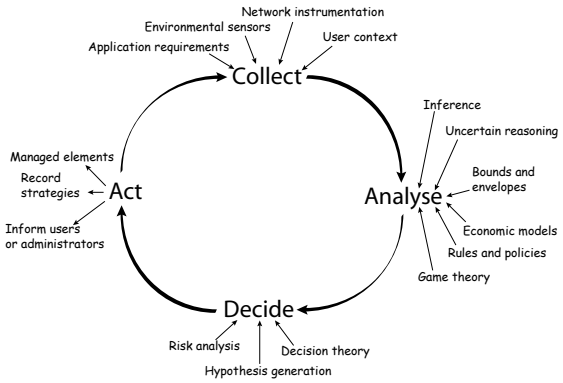

Figure 1. The autonomic control loop (from [3])

to understand the system as a whole, and how changes in components affect its overall behaviour.

3 Interactions as an object of study

How might we come to such an understanding? It is clearly a very complex challenge, but one way to approach it is to explore what properties we would like to see in such a model: what features would make it easy to study the interactions between components, and to prove properties about them? What follows is a first attempt to sketch out such a feature list.

3.1 Providing envelopes of behaviour

A user's perception of a system is based on the system itself, not on its components. The internal structure is typically of interest to designers and maintainers, but not to users. Therefore our first desirable property is that we are able to describe the behaviour of the system as a whole.

This is more of a challenge than it might first appear. The behaviour of individual components may affect those of others, and their individual adaptations may change these effects over time. We may not be able to say exactly what a system will do at any point. As a concrete example, consider a video-on-demand streamer whose network driver adapts to changing traffic conditions. The system may only be able to deliver restricted frame rates at times of high load, or may drop frames or reduce resolution. Predicting exactly what a user will experience at any time involves knowing the network environment. However, we can still make meaningful statements about system behaviour. We might, for example, say that it is better to stop a stream entirely rather than allow it to be seen degraded beyond a certain critical threshold. Another strategy might be to time-shift the stream, downloading it over time and then playing it locally rather than streaming it over the congested network.

Such statements establish an *envelope of behaviour* for the video streamer. We cannot say what it will do in detail at any time, but we *can* say that it will remain within a particular envelope of user quality. Note that this says little about the individual components of the streamer: rather, it allows free rein for the designer to develop optimised protocols and so forth, as long as these techniques can be co-ordinated to keep the system within envelope.

While we tend to focus on *changes* in behaviour, it is perhaps equally important that we understand those areas in which behaviour *stays the same*: where different organisations and approaches can yield the same behaviour (for example see [1]), or where we want behaviour to remain constant even to the detriment of other factors.

3.2 The meaning of systems

When we suggested above that one way to deal with network congestion was to time-shift the video and deliver it from local storage, we were stepping well beyond the traditional framework of protocol design. In essence we were making use of knowledge that we have about the ways in which people consume video: that they might prefer a good experience a little later than a degraded experience when they ask for it. The strategy we chose is *only* valid under some specific conditions of user activity. This means that the relationship of the streamer to the network is being conditioned by the use that will be made of the stream at the end-point.

Traditional communications design does not work like this: it moves uninterpreted bit streams across the network, under constraints of bandwidth, jitter and so forth. However, knowing the *meaning* of the bit stream might change the constraints: a video intended for immediate viewing (streaming) has different constraints to the same video being intended for later viewing (downloading) [2]. The *type* of the data is not the critical factor: rather, the *semantics of*

the connection is derived from how the information will be used.

Of course this distinction is already made to some degree: we might use RSVP or RTSP to stream a video, whilst using HTTP or FTP to download it. The point is that this is not a *network* decision, or one that can be made within a single component, but one that relies on higher-level knowledge. If we capture this knowledge into the system we can make such decisions internally.

3.3 Supporting inputs and context

Pervasive computing relies a lot on context – often defined as the environment in which activity occurs, understood symbolically. A pervasive system provides service to users, but those services and their delivery may adapt to the users' location, devices, tasks, constraints and other factors.

Such a system will have two distinct forms of input: the "classical" inputs such as keystrokes and gestures, together with "contextual" inputs derived from the environment. Since changes in either can affect the users' experience, it makes little sense to handle them differently at a model level.

For an autonomic system, we need to interpret context more widely than is usual for a pervasive system. As well as the user-centric context of location, device, task and so on, we might treat network latency, bandwidth, contention, the presence of firewalls and other internal factors as context which affects the system's behaviour.

Introducing complex context can be daunting, since it increases the number of degrees of freedom that a designer must understand. There will however often be additional (and sometimes quite rich) structure within both context and the behaviours and adaptations that it induces [4], and these may be leveraged. Such structure can be quite effective in helping to define the envelope of behaviour of a system.

3.4 Principled trade-offs

When we speak of optimising behaviour, it is important to remember that we optimise against specific criteria. This implies that we cannot optimise against everything, which in turn implies that optimising a system along one dimension may – and typically will – result in sub-optimal behaviour on another dimension.

How can a designer decide which dimensions are worth optimising? This decision is typically easier with respect to components than with respect to systems: a network component might optimise throughput, but if this compromises (for example) the system's ability to satisfy multiple users simultaneously then it was a poor choice. Moreover the "correct" choice may change over time and with context.

A design model may not be able to help a designer make this choice, but it *may* enable her to understand the implications of different choices and how they impact on the system's overall behaviour by illuminating how the choices interact with the envelope of behaviour or the semantics of actions.

3.5 Evolution

An autonomic system designer's work is never done: any interesting, long-lived system will exhibit a constant process of change in terms of its components but also in its tasks and constraints. A system design must be "living", in the sense that it can continue to track and reflect the changes that must be made as the system evolves.

Perhaps a good analogy is with "sound trip engineering" tools for software engineering, in which changes in a design document (typically one or more UML diagrams) are automatically reflected by changes in the associated code, and *vice versa*. This is not to say that round-trip engineering is a reasonable target, and one may criticise such tools as destroying much of the richness of a design by forcing conceptually different structures into the same code patterns. The point is that design, its artifacts and the associated analyses and decisions are part of an on-going process – and *it is the process that is important*, not the artifacts.

4 Conclusion

In this short paper we have argued for a more holistic, more principled and more structured approach to the design and engineering of autonomic systems.

There are a number of strands of research that are developing analysis and programming techniques to deal with the issues raised here, and we are grateful to the authors of this research for their inspiration. We do not seek here to argue for one approach over another: the important point is to encourage a change in the ways in which we conduct the design processes for complex adaptive systems. We firmly believe that a focus on individual problems in isolation is rapidly becoming a hindrance to the effective construction of systems.

This viewpoint has wider ramifications. Like any discipline, software engineering has seen a massive increase in specialisation, and specialisation is harmful to developing a balanced, holistic view of a large system. By providing common, broad-spectrum models within which to develop, compare and analyse different parts of a system, we allow specialists to see how their activities impact in the wider systems context, without unnecessarily prioritising any particular aspect of a design ahead of time and without fossilising design decisions too early in the process.

Acknowledgements

This article is the IEEE Computer Society – Task Force on Autonomous and Autonomic Systems Oct/Nov 2006 – Letter released through the TFAAS Newsletter – Issue 7 @ http://tab.computer.org/aas

The article is based on a keynote talk given by the author at IFIP Autonomic Networking, Paris FR, September 2006.

References

[1] V. de Silva and R. Ghrist. Homological sensor networks. *Notices of the American Mathematical Society*, 54(1):10–17, 2007.

[2] S. Dobson. Putting meaning into the network: some semantic issues for the design of autonomic communications systems. In M. Smirnov, editor, *Proceedings of the 1st IFIP Workshop on Autonomic Communications*, volume 3457 of *LNCS*, pages 207–216. Springer Verlag, 2005.

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

[4] S. Dobson and P. Nixon. More principled design of pervasive computing systems. In R. Bastide and J. Roth, editors, *Human computer interaction and interactive systems*, volume 3425 of *LNCS*. Springer Verlag, 2004.

[5] E. Gelenbe and R. Lent. Power-aware *ad hoc* cognitive packet networks. *Ad Hoc Networks Journal*, 2(3):205–216, 2004.

[6] J. Kephart and D. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–52, January 2003.

[7] M. Razzaque, P. Nixon, and S. Dobson. Demonstrating the feasibility of an autonomic communications-targeted cross-layer architecture. In *Proceedings of the 14th International Conference on Advanced Computing and Communications*, 2006.