When to Adapt? Identification of Problem Domains for Adaptive Systems

Kristopher Welsh and Pete Sawyer

Lancaster University, Computing Dept., Infolab21 LA1 4WA Lancaster, UK
{k.welsh,p.sawyer}@lancs.ac.uk

Abstract. Dynamically adaptive systems (DASs) change behaviour at run-time to operate in volatile environments. As we learn how best to design and build systems with greater autonomy, we must also consider when to do so. Thus far, DASs have tended to showcase the benefits of adaptation infrastructures with little understanding of what characterizes the problem domains that require run-time adaptation. This position paper posits that context-dependent variation in the acceptable trade-offs between non-functional requirements is a key indicator of problems that require dynamically adaptive solutions.

Keywords: Adaptive systems, non-functional requirements.

1 Introduction

Kephart and Chess [1] identified the move to autonomic computing as a grand challenge to the software engineering community. They argue that systems able to monitor, (re)configure, (re)construct, heal and tune themselves at run-time, are needed to mitigate the ever increasing size and complexity of computing systems; which are expected to operate in ever less predictable and stable environments. Although such systems remain out of reach today, important steps toward them are being taken by the research community with self-managed, or dynamically adaptive systems (DASs). A DAS alters its behaviour or composition in response to changes in its environment.

All software systems have to cope with changes in their environment, but usually the environment changes slowly enough for adaptation to be performed off-line. Web browsers, for example, need to adapt to cope with new content types and protocols with the development of new versions that can be installed as updates on users' computers. Increasingly, however, systems are being conceived that need to adapt at run-time. For example, applications at the "wireless edge" of the Internet, must adapt to the fluctuating availability of services as users move between areas covered by different networks. Other examples include systems adapting to cope with different user needs [2], new network topologies [3][4], and radical change in physical environments [5].

There are two common types of adaptation: parametric and architectural [6]. Parametric adaptation involves building adaptive capabilities into code on a per-application basis, radically increasing complexity and making DASs costly to build and maintain. Architectural adaptation, by contrast, uses an adaptive infrastructure

B. Paech et al. (Eds.): REFSQ 2008, LNCS 5025, pp. 198–203, 2008.

which typically effects adaptation by component substitution without suspending execution [7][8]. Adaptive complexity is partitioned to a reusable and configurable adaptive infrastructure, easing the maintenance of applications that use it. Dynamic adaptation is a technology that is still maturing and many of the DASs reported in the literature have been developed to showcase the capabilities of particular adaptation infrastructures.

Although adaptive infrastructures provide a mechanism for easing implementation complexity at the implementation level, the complexity inherent in the problems for which DASs provide a solution remains a challenge. As the enabling technology continues to mature, we will need to improve our understanding of how to analyse, specify and design DASs, so that we can cope with the conceptual complexity posed by volatile environments. At the requirements level, Berry et. al. [9] have identified four levels of RE needed for DASs, which has been used as the basis for subsequent work on goal-driven analysis of DASs [10][11], along with other approaches investigating their requirements: e.g. [12]. In most cases, the RE for DASs start with an assumption that the problem under analysis requires a DAS as the solution, and that therefore the need for dynamic adaptation is somehow obvious from the outset. There may well be families of problems where this will be true, but it may not always be clear. Such ambiguity risks over-engineering systems for which dynamic adaptation is not, in fact, a requirement. Similarly, failure to recognize the presence of such a requirement early in a project may result in cost underestimation or worse.

In this position paper, we posit that a problem that requires a DAS will exhibit identifiable characteristics that if not present, strongly indicate that a conventional, static system will provide an adequate solution. If our hypothesis holds true, it should act as a litmus test usable for analysts during the early-phases of RE.

2 Volatile Problem Domains, Adaptive Requirements

For our purposes here, we consider the requirement to adapt dynamically to be imposed by the environment in which the system must operate. In general, we exclude systems that use adaptation as a defensive strategy to cope with design or implementation failures, perhaps by adopting a 'limp-home' mode on detection of a failed component. An exception to this rule is where the system is designed to cope with failure conditions that have their root in a 'failure' of the analysis process to anticipate possible states of the environment. In our terms, dynamic adaptation is a legitimate mitigating strategy when the analyst recognizes that their model of the environment is incomplete. For example, there may unknowable properties of the atmosphere of Mars that the designers of a probe nevertheless need to try to cope with.

Systems that must cope with unknowable environments are at the extreme end of a spectrum of DASs. More common is the situation where the environment is volatile but understood sufficiently well to allow the analyst to anticipate how it will change. Here, the approach advocated by Berry et al [9] is to characterize the environment as a set of discrete stable *domains* that it can transition between. A DAS can then be conceptualized to comprise a set of *target systems*, each designed to operate within a domain. The analyst's job is then to specify each target system and the *adaptation*

scenarios [10][11][13] that specify when the system adapts from one target system to another.

The question we seek to answer is how can a requirement for dynamic adaptation be identified early in the development process? This can be re-phrased as what features of the problem domain indicate that a DAS will provide an appropriate solution? There are two non-exclusive classes of environment which imply a need for dynamic adaptability. The first class is where the requirements that are consequent on the environment change on a time-scale likely to be experienced by the running system. For example, a mobile device may need the ability to adapt in order to take advantage of new services as they come in range and become available. The second class is where the trade-offs between non-functional requirements (NFRs) varies with context. Here, the set of requirements may be constant, but what constitutes their satisfaction is not. We hypothesize that the second class is the more common but also more subtle and harder to recognise. For example, in the case of the mobile device above, the choice of service to use may be constrained by a preference for certain service providers that may not always be available.

In the next section we examine two examples of DASs to illustrate that NFR trade-offs are a common feature of each. By so doing they provide evidence in support of our hypothesis.

3 DAS Exemplars

Our first example DAS is an image viewer that adapts to usage patterns and available resources. The system, presented in [4] loads images either from the local file system or a remote URL, and caches images to reduce latency when there is sufficient memory available. Although no requirements process is reported for the system, it is trivial to elicit the two primary non-functional requirements that the adaptation addresses: minimise latency when switching between images and minimise memory usage to avoid swapping. The time taken to load images from local and remote file systems is variable, as is the amount of memory available.

During normal operation, the “minimise memory usage” NFR is given priority, with the system performing no caching. However, when image loading and decoding time exceeds a given threshold, the system adds a caching component, satisficing the “minimise latency” NFR. The viewer also monitors free memory, disabling the cache when scarce and using parametric adaptation to adjust cache size during operation.

Parametric adaptation is also used to switch cache replacement policy: selecting a Most Recently Used policy if images are accessed sequentially, and a Least Recently Used policy otherwise. This essentially tunes the system to best satisfy the “minimise latency” NFR according to usage. They key point is that what constitutes satisfaction of the NFRs varies with the operating environment, thus making adaptation advantageous.

Our second example is an adaptive flood prediction and monitoring system deployed on the banks of the river Ribble in North West England [5]. *GridStix* is an intelligent wireless sensor network that monitors the river and analyses data gathered by multiple sensor nodes. The sensor nodes have enough processing power to process the data co-operatively by acting as a lightweight computational grid, obviating the need to transmit raw water depth and flow rate data off-site for processing. This is significant because GridStix’s remote location means that only low-bandwidth

cellular radio networks are available for long-range data transmission. The remote location also means that GridStix is dependent on batteries and solar panels for its power supply. Another feature mandated for GridStix is the use of digital camera images for flow sensing. Digital cameras are inexpensive and robust but produce large volumes of data. The ability to process this data locally is a precondition for the use of digicams.

GridStix's environment has been characterized by domain experts according to three distinct *domains*. In the first the river is *quiescent*. In the second domain, *high flow*, the river flows rapidly but still without significant depth increase. A high flow rate can presage the arrival of a pulse of water that would result in the third domain, *flood*, where both the flow rate and the depth are high. GridStix's key NFRs are "energy efficiency" to maximise battery life, "accuracy" to provide timely and accurate flood warnings, and "fault tolerance" to aid survivability. Crucially, the relative importance of the NFRs varies with the domain. In the quiescent domain, energy efficiency has the priority. With no flood event imminent, the key requirement is to keep the system in readiness, sampling data relatively infrequently. In the high flow domain, the possibility of the onset of a flood event means that accuracy of prediction is relatively more important than it is in the quiescent domain. This means that sampling needs to happen more frequently and the data needs to be processed more quickly. In the flood domain, GridStix still needs to provide accurate predictions but the ability to survive node loss due to submersion or water-borne debris promotes the relative importance of fault-tolerance.

GridStix needs to adapt to the three domains to ensure the appropriate trade-offs between the three NFRs. A reflective middleware platform supports this by, for example, substituting components for different spanning tree algorithms that enable the sensor nodes to communicate. A relatively energy-efficient shortest-path algorithm is used for the quiescent and high flow domains. A more power-hungry but resilient fewest-hop algorithm is used for the flood domain.

Many flood warning systems use sensor networks. Most of these are 'dumb', with no grid-like computational capability. This precludes, for example the use of inexpensive digital camera imaging for flow sensing since the volumes of data are too high to transmit off-site for processing over low-bandwidth communication networks. Nevertheless, such systems are subject to many of the same NFRs as GridStix. Satisfaction of both the fault-tolerance and energy-efficiency requirements is significantly inhibited, however, if the systems are unable to adapt as their river environments change. Hence, although flood warning systems need not necessarily be DASs, the peculiar combination of NFRs to which they are subject make a strong case for them being implemented as DASs. The same argument can be made in many other domains where dynamic adaptability offers better solutions than have hitherto been available.

Both our exemplars exhibit environment volatility. The image processing system has to cope with network latency, while the flood warning system has to cope with a river subject to frequent heavy rainfall. In both cases, the key goals of the system remain the same irrespective of the environment; to render images and to predict flooding, respectively. In both cases, however, the acceptable trade off between their NFRs varies. We hypothesise that this NFR trade-off characteristic is a key signifier that dynamic adaptation is needed.

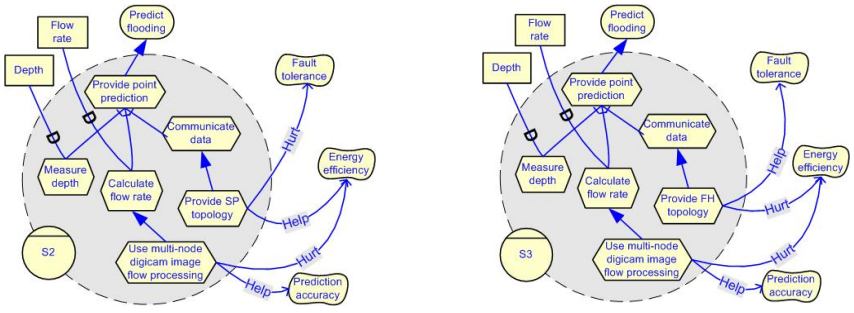

Fig. 1. Models of GridStix configured for *High Flow (S2)* and *Flood (S3)* domains

In [10] we have proposed the use of *i** [14] for making the trade-offs between NFRs explicit. Figure 1 illustrates this by showing developments of two models of how GridStix is configured for the High Flow and Flood domains. The key features are the three softgoals representing the NFRs on the right of each part of the figure. Notice how Fault tolerance and Energy efficiency are either helped or hurt by substituting the task *Provide FH (fewest hop) Topology* for the *Provide SP (shortest path) Topology* as the system adapts from High Flow to Flood. In our approach, which follows closely the three levels of RE for DASs proposed by Berry et al. [9], the models in Figure 1 are developed following development of a strategic dependency graph that models in which the overall goals and softgoals are identified. Subsequent models are developed to specify the adaptation scenarios and to inform the selection of the adaptive infrastructure.

4 Conclusion

Dynamic adaptation allows us to create systems able to operate in environments that have hitherto posed daunting problems for system developers. As ubiquitous computing begins to demand greater context-awareness and flexibility we will encounter problem domains requiring dynamic adaptation increasingly often. Since adaptive systems are fundamentally more complex than static systems, however, being able to identify such problems early on in the RE process is important.

There currently exists no systematic means to recognize the characteristics of a problem that requires a dynamically adaptive solution. Great advances have been made in the development of adaptive infrastructures but the RE community has been slow to respond to the challenges posed by the kinds of problem that adaptive infrastructures are designed to support. The RE community is now beginning to show some awareness, as evidenced by, for example [2] [9] [11].

Our aim in writing this paper has been to argue that a key capability of RE is early recognition of whether a problem demands a dynamically adaptive solution. We have not shown that this can be done in all cases. Rather, we have posited the idea that where analysis of a problem identifies a set of NFRs whose relative priorities change according to the state of the environment, a capability for dynamic adaptability *may*

be a key requirement of the solution. Two exemplars have illustrated our ideas. We now need to test our hypothesis in a wider range of applications to see whether our hypothesis holds. If it does hold, then we will have a useful litmus test of one aspect of complexity that impacts significantly on system development.

References

1. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1) (2003)
2. Fickas, S.: Clinical requirements engineering. In: Proceedings of the 27th International Conference on Software engineering (2005)
3. Cerpa, A., Estrin, D.: ASCENT: adaptive self-configuring sensor networks topologies. Transactions on Mobile Computing 3(3) (2004)
4. David, P.C., Ledoux, T.: Towards a Framework for Self-Adaptive Component-Based Applications. In: Stefani, J.-B., Demeure, I., Hagimont, D. (eds.) DAIS 2003. LNCS, vol. 2893, pp. 1–14. Springer, Heidelberg (2003)
5. Hughes, D., Greenwood, P., Coulson, G., Blair, G.: GridStix: supporting flood prediction using embedded hardware and next generation grid middleware. World of Wireless, Mobile and Multimedia Networks (2006)
6. Mckinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.C.: Composing adaptive software. IEEE Computer 37(7) (2004)
7. Kramer, J., Magee, J.: Self-Managed Systems: an Architectural Challenge. Future of Software Engineering (2007)
8. Karsai, G., Ledeczi, A., Sztipanovits, J., Peceli, G., Simon, G.,, Kovacshazy, T.: An Approach to Self-adaptive Software Based on Supervisory Control. In: Laddaga, R., Shrobe, H.E., Robertson, P. (eds.) IWSAS 2001. LNCS, vol. 2614, pp. 24–38. Springer, Heidelberg (2003)
9. Berry, D.M., Cheng, B.H., Zhang, J.: The four levels of requirements engineering for and in dynamic adaptive systems. In: Proc. 11th International Workshop on Requirements Engineering: Foundation for Software Quality, Porto, Portugal (2005)
10. Sawyer, P., Bencomo, N., Hughes, D., Grace, P., Goldsby, H., Cheng, B.: Visualizing the Analysis of Dynamically Adaptive Systems Using i* and DSLs. In: Proc. 2nd Intl. Workshop on Requirements Engineering Visualization, Delhi, India (2007)
11. Goldsby, H., Cheng, B.H.C.: Goal-Oriented Modeling of Requirements Engineering for Dynamically Adaptive System. In: Proc. 14th IEEE International Requirements Engineering Conference, Minneapolis, USA (2006)
12. Sora, I., Cretu, V., Verbaeten, P., Berbers, Y.: Managing Variability of Self-customizable Systems through Composable Components. Software Process: Improvement and Practice 10(1) (2005)
13. Efstratiou, C., Cheverst, K., Davies, N., Friday, A.: An Architecture for the Effective Support of Adaptive Context-Aware Applications. In: Proc. Second International Conference on Mobile Data Management, Hong Kong (2001)
14. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: 3rd IEEE Int. Symp. on Requirements Engineering (RE 1997), Washington D.C., USA (1997)