Challenges of Highly Adaptable Information Systems

Stephen Cook, Rachel Harrison, Timothy Millea & Lily Sun
Applied Software Engineering Research Group
University of Reading

18th August 2003

Keywords: architecture description, autonomic computing, design pattern language, e-learning, information system architecture, simulation, software evolution, system dynamics

1 Introduction

The success of personal, networked computing (most obviously in the form of the World Wide Web) has encouraged computerisation in application domains that were previously found (or assumed) to be unsuitable for it. Some of these domains are characterised by:

- imprecise and volatile requirements;
- frequent reconfigurations of processes, strategies and objectives;
- complex rules with innumerable exceptions;
- high (and often rising) user expectations (e.g. for usability, customisation).

In other words, information systems in these domains must be highly adaptable if they are to satisfy users' complex and rapidly evolving requirements.

This position paper identifies four current research areas in software engineering that are critical success factors for the development of highly adaptable information systems. The e-learning domain is used as a running example. Section 2 introduces some background material and related work in the areas of information system architecture, software evolution and e-learning. Section 3 outlines the issues that define this research programme. Section 4 relates these concerns to current research in the Applied Software Engineering Research Group at the University of Reading.

2 Background and Related Work

2.1 Architecture and evolution in information systems

The architecture and the evolution of any information system are closely related, as illustrated by their definitions. IEEE Standard 1471-2000 defines architecture as:

> 'The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.' [12]

The phenomenom of software evolution, first identified by Lehman and Belady [15], refers to a process of continual change in software systems, particularly in the growth of their functionality and complexity. Implicitly, each incremental step in the evolution of a software system involves the adaptation of some of its architectural properties and the preservation of others.

The architectural properties of any particular system are not equally adaptable. Some may be so difficult (and therefore expensive) to change, that they are effectively invariants of a system, or of a product line of systems, or even of an entire enterprise. This kind of architectural property can be thought of as an investment that is intended, in part, to reduce the costs of future adaptations to the system. (Arguably, the return on investment of architectural work should be measured in terms of reducing the system's maintenance costs.) However, over the lifetime of a system, its evolution may expose either weaknesses or inflexibility in its architecture (especially if the evolution was unanticipated), which in turn may raise the costs of ownership and even threaten the viability of the system.

The behaviour of these relationships is not yet well understood; case studies of real-world domains that require highly adaptable information systems can improve our knowledge.

2.2 Flexible e-learning systems

Computer-based training (CBT) and distance-learning are well-established, niche alternatives to traditional ('chalk-and-talk') models of education. The e-learning [19] concept builds on these traditions but also adds powerful new ingredients drawn from network-centric computing, computer-supported co-operative work (CSCW), adaptive environments, flexible processes and component-based software reuse. Effectively, e-learning may be regarded as a new paradigm of education that could improve flexibility, quality and participation in education and training [17, 13]. Ambitious plans are already being made for e-learning to play a major role in expanding higher education [5]; the University of Reading is directly involved in these initiatives through its leadership of the Thames Valley New Technology Institute[1].

Considerable resources have been applied to e-learning by industrial trainers, educational institutes and software producers. Several COTS products (e.g. Lotus LearningSpace[2], Blackboard[3], Oracle iLearning[4]) are available and have established a baseline of functionality that enables a tutor to publish teaching materials online, create discussion forums, organise assessments, and link to other resources [2, 10].

However current e-learning products have been less successful so far in providing more advanced functionality. For example, although many learning systems can provide simple customisation (e.g. a choice of font families), richer forms of personalisation currently depend on personal, usually face-to-face, interaction between teacher and learner. Consequently, if current e-learning systems were to simply replace traditional educational models, there would be a significant risk that the quality of the learning experience would deteriorate.

The challenges facing the next generation of e-learning systems include the provision of:

- improved ability to adapt rapidly and transparently to changes in a learner's profile and his/her progress through a learning package;
- better mechanisms for discovering and comparing relevant learning resources;
- the ability to specify the requirements of an instructional component and delegate the discovery of a resource that satisfies it to another process (e.g. a software agent);
- low-maintenance systems that are easy to inter-operate with both external resources and other education management systems.

This implies that e-learning systems face the challenge of how to evolve rapidly to become semantically rich, highly dynamic, distributed and personalised to the needs of individual users.

3 Architectural Challenges of Highly Adaptable Systems

3.1 Assessing information system evolvability

The IEEE definition of architecture cited in section 2.1 assigns a major role to architecture in defining the evolutionary principles of software-intensive systems. This role is poorly supported by existing modelling languages and tools, which tend to focus on system structure, operational behaviour and communications. However, the architectural properties of a system have to be considered at various levels of abstraction, from policies and principles (the 'Contextual' level) down to servers and programs (the 'Components' level) [20]. Consequently, architects and other stakeholders are often hampered in assessing whether the architecture of a system supports its expected evolution across the range of their concerns.

Some support for assessing architectural adaptability is provided by scenario-based methods (e.g. ATAM [8]). The explicit identification of architectural commonalities and variability has been recognised as particularly important in product-line engineering [9]. However, these approaches may not

[1] http://www.hefce.ac.uk/News/hefce/2002/NTIs.htm
[2] http://www.lotus.com/products/learnspace.nsf/wdocs/
[3] http://www.blackboard.com/
[4] http://ilearning.oracle.com/

scale up gracefully in domains such as e-learning that are characterised by complex and rapidly changing concepts. The variabilities in, for example, 'teaching resources' cover a potentially vast range of cross-cutting concerns (e.g. teaching methods and technologies, language and culture of learning milieu, students' level of education and prior experience, applied *vs.* theoretical focus of course). Compared with physical products, it is much more difficult to identify a stable, core 'chassis' that could be adapted using standardised, bolt-on components.

3.2 Architectures for low-maintenance information systems

Highly adaptable information systems are implicitly expected to adapt intelligently to a continuous stream of events, both from within the system and from its environment. Currently, complex adaptations of software usually require manual intervention by skilled (hence expensive and often scarce) personnel. Unless intelligent adaptive processes can be largely automated, it will be impossible to prevent highly adaptable systems from becoming 'support-bound' as they increase in scale.

3.3 Using design patterns in rapidly evolving domains

The design of highly adaptable systems should make use of design patterns [11] to achieve the following benefits:

- simplified software maintenance (assumption: the explicit use of well-known design patterns makes systems easier to understand);
- more adaptable systems (many 'classic' design patterns are directed at solving problems of system evolution).

However, it is unclear how design patterns should be used in domains that are evolving rapidly. For example, some approaches (e.g. [4]) have chosen to add explicit and detailed domain knowledge to individual patterns but to leave implicit any relationship to 'deeper', domain-independent patterns such as those catalogued in [11]. This approach could lead to inflexible, rather than evolvable, designs if there is a high risk that the knowledge will be modified in the future.

3.4 Assessing the dynamics of architecturally complex systems

It will not be possible to accurately predict either the dynamic behaviour or the evolution of highly adaptable systems by purely static analysis of their programs; some form of behavioural modelling or simulation will be essential. These models will need to take account of:

- the technological environment in which the system operates;
- the social and business processes that the system is intended to support;
- the 'global software process' [16] in which the evolution of the system is managed.

Models will also need to take account of a wide range of timescales, from very short-term (as services vary dynamically during a user's online session) to much longer-term (as services, agents and resources evolve through both technological and business life cycles).

4 Proposed Research Programme

4.1 Architecture description languages for evolutionary properties

A case study of the e-learning domain can explore practical approaches to assessing system evolvability (section 3.1) by investigating:

- which concepts of evolution are most relevant to highly adaptable information systems,
- how the concepts could be represented as a simple grammar, and
- how to anchor them to software engineering theory.

The results could assist the design of a structured language for unambiguously describing the evolutionary requirements and capabilities of a system in architectural terms (i.e. defining an evolution *viewpoint* and its *model*, to use the terminology of IEEE 1471-2000 [12]).

4.2 Architectures for autonomic information system services

One approach that could mitigate the risk of highly adaptable systems becoming support-bound (section 3.2) is autonomic computing. The term autonomic takes its meaning from the self-regulation of the central nervous system, in which functions such as heart beat rate, blood sugar levels and perspiration are adjusted without conscious thought and according to changing external conditions. By analogy, autonomic computing systems should regulate and maintain themselves to provide an optimal level of service without the conscious intervention of either the user or maintenance staff.

The e-learning domain provides an opportunity to assess the emerging results of our 'Autonomic Computing – Creating self-Evolving Software Systems' (ACCESS) project. ACCESS[5] introduces a model in which the evolution of a software system is guided by resolving the expressed concerns of its stakeholders. The resolution process operates within a space of possibilities defined by a software component market. This approach to automated 'just-in-time' system evolution develops ideas on ultra-rapid evolution that were proposed by Bennett *et al.* [6]. A schematic diagram of ACCESS's proposed architecture is shown in figure 1.

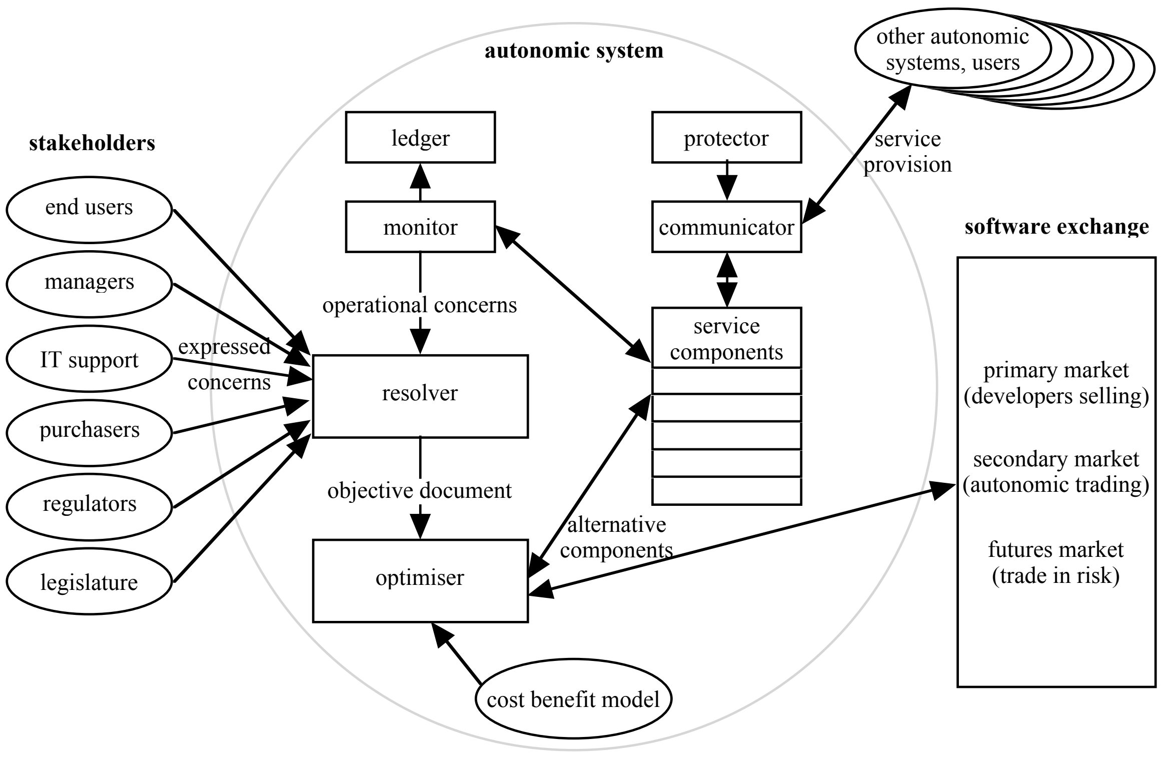

Figure 1: ACCESS schematic architecture

The ACCESS approach is by intention domain-independent and its architecture is highly abstract. Applying it to a specific rapidly evolving domain such as e-learning will raise many questions, including:

- is a market-based metaphor suitable for an activity such as education that has multiple, potentially conflicting, objectives?
- most competitive social situations, including markets, require a regulatory function that is independent of the broking function (i.e. the 'autonomic system' in figure 1); how should this be provided in an e-learning context?
- fairness in market-based allocation systems depends critically on all stakeholders having similar access to reliable information, but feedback to stakeholders is only implicit in the ACCESS

[5] EPSRC grant no. GR/S19066/01

architecture; what additional, possibly domain-specific, feedback channels are needed to ensure fair access for stakeholders to information?

4.3 Design pattern languages for rapidly evolving domains

The issue of how to relate design patterns to domain knowledge (section 3.3) can be addressed by investigating whether the concepts of system evolution provide an effective rationale for structuring design patterns in rapidly evolving domains. For example, Simon [18] suggested that the qualities of *hierarchical* and *nearly decomposable* organisation make it easier for a system to evolve. This may imply that when systems are required to be highly adaptable, their atomic design patterns should be as domain-independent as possible, and the binding to a specific domain should be achieved at a higher level, i.e. through an arrangment of selected patterns into a *pattern language* [1].

This approach could be seen as a generalisation of the coordination patterns that Andrade *et al.* [3] proposed as a mechanism for allowing business rules to evolve independently of core business concepts. It is also implicitly related to Lehman's SPE taxonomy [14]; the concept of patterns as reusable, domain-independent solutions seems similar to Lehman's *P-type* components (which are less likely to evolve), while pattern languages seem closer to his *E-type* components (which inevitably evolve).

Case studies and experiments are needed to explore which of these concepts are both relevant and scalable to the demands of e-learning systems. The development of new IT courses at Reading University provides an opportunity to conduct pilot studies, e.g. to compare the effectiveness of different approaches to the design and use of pattern languages for the e-learning domain.

4.4 Simulation of architectural evolution

Previous simulation studies of software evolution (e.g. [7]) have usually treated a software system as a black-box component and have not attempted to consider the effects of the system's architecture. On the other hand, simulation models of computer networks do take account of network architecture but often model the architecture as a simple, recursive structure. These approaches, if taken separately, may not be sufficient to produce accurate predictions of the dynamics of highly adaptable systems on either short- or long-term timescales.

One of the questions that we plan to investigate is whether models of software evolution can be improved by introducing selected information about the system's architecture. For example, referring again to Lehman's SPE classification of software components, does knowing the proportions of *E-* and *P-type* components in a system improve predictions of the course of its evolution? The ultimate goal would be to discover which architectural properties (i.e. fundamental design choices) of highly adaptable systems are most important in determining the shape of a system's subsequent evolution.

5 Conclusions

The demands of highly adaptable information systems provide a challenge for many aspects of software engineering, especially those related to architecture and evolution. The e-learning domain is very suitable for investigating these problems because it is entering a phase of rapid change. Furthermore, this rapidly evolving domain has a direct impact on many higher education institutions, which creates opportunities for researchers to also explore the practicality of candidate solutions to the problems that this paper has identified.

References

[1] Alexander, C., Ishikawa, S. and Silverstein, M., 1977. *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press.

[2] Anderson, M.D., 1997. Critical elements of an Internet based asynchronous distance education course. *Journal of Educational Technology Systems,* **26**(4), 383–388.

[3] Andrade, L., Fiadeiro, J., *et al.*, 2000. Patterns for coordination. *In*: Catalin-Roman, G. and Porto, A., ed. *Coordination Languages and Models.* Springer-Verlag (Lecture Notes in Computer Science, 1906), 317–322.

[4] Avgeriou, P., Papasalouros, A., *et al.*, 2003. Towards a pattern language for Learning Management Systems. *Educational Technology & Society,* **6**(2), 11–24.

[5] Beller, M. and Or, E., 1998. The crossroads between lifelong learning and information technology: a challenge facing leading universities. *Journal of Computer Mediated Communication,* **4**(2) December, .

[6] Bennett, K., Munro, M., *et al.*, 2001. An architectural model for service-based software with ultra rapid evolution. *In*: *Proceedings of the IEEE International Conference On Software Maintenance (ICSM 2001): Systems and Software Evolution in the Era of the Internet*, Florence, Italy, 7–9 November 2001. Los Alamitos, CA: IEEE Computer Society, 292–300.

[7] Chatters, B.W., Lehman, M.M., *et al.*, 2000. Modelling a software evolution process: a long-term case study. *Journal of Software Process: Improvement and Practice,* **5**(2–3), 95–102.

[8] Clements, P., Kazman, R. and Klein, M., 2002. *Evaluating Software Architectures: Methods and Case Studies.* Boston, MA: Addison- Wesley (Software Engineering Institute series).

[9] Coplien, J., Hoffman, D. and Weiss, D., 1998. Commonality and variability in software engineering. *IEEE Software,* **15**(6) November/December, 37–45.

[10] El-Tigi, M. and Branch, R.M., 1997. Designing for interaction, learner control, and feedback during Web-based learning. *Educational Technology,* **37**(3), 23–29.

[11] Gamma, E., Helm, R., *et al.*, 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston, MA: Addison-Wesley (Professional Computing series).

[12] IEEE Computer Society, 2000. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE-Std-1471- 2000. New York: IEEE.

[13] Learning Systems Architecture Lab, 2002. *SCORM Best Practices Guide for Content Developers.* Pittsburgh, PA: Carnegie Mellon University.

[14] Lehman, M.M., 1980. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE,* **68**(9), 1060–1076.

[15] Lehman, M.M. and Belady, L.A. (eds.), 1985. *Program Evolution: Processes of Software Change.* London: Academic Press (A.P.I.C. Studies in Data Processing, 27).

[16] Lehman, M.M. and Kahen, G., 2000. A brief review of feedback dimensions in the global software process. *In*: Ramil, J.F., ed. *FEAST 2000 Workshop: Feedback and Evolution in Software and Business Processes*, London, UK, 10–12 July 2000. London: Imperial College of Science, Technology and Medicine, 44–49.

[17] Schweizer, H., 1999. *Designing and Teaching an Online Course: Spinning Your Web Classroom.* Needham Heights, MA: Allyn and Bacon.

[18] Simon, H.A., 1969. *The Sciences of the Artificial.* Cambridge, MA: M.I.T. Press.

[19] Sloman, M., 2001. *The E-Learning Revolution: From Propositions to Reality.* London: CIPD.

[20] Zachman, J.A., 1987. A framework for information systems architecture. *IBM Systems Journal,* **26**(3), 276–292.