ADAPTOR: a pattern language for the re-engineering of systems to Object Technology

Alan O’Callaghan
Department of Computer Science
Faculty of Computing Sciences and Engineering
De Montfort University
The Gateway
LEICESTER LE1 9BH
aoc@dmu.ac.uk

In this paper the burning issue of the migration of legacy systems to object technology is examined and the major elements of an evolving pattern language, ADAPTOR (Architecture-Driven And Pattern-based Techniques for Object Re-engineering) to illustrate an approach currently being investigated by the author. ADAPTOR can be considered to be one of a set of interlinked pattern languages that address the wider issues of software architecture. It is given primacy here because of its relative maturity, having been tested in a number of live projects in the telecommunications industry.

1. Introduction

Software architecture is a relatively new field of study, the importance of which is growing steadily as software practitioners, both in industry and academia, have recognised its significance to the building of robust, long-lived, and yet flexible software systems (which can be single, large-scale applications or families of applications). According to Bass et al., “A software architecture is the development product that gives the highest return on investment with respect to quality, schedule and cost” [Bass 1997, p.x]. However, there is no consensus on how best to describe architectures [Northrop 1997], and no established development practice for software architecture as yet. Indeed, there is no single definition that has yet emerged which can help to unify practices and ideas. Linda Northrup has most recently provided the following notion,

“...The software architecture of a program or computing system is the structure or structures of the system, which compromise software components, the externally visible properties of those components, and the relationship among them” [Northrup 1997]

At the same time as this growth in interest in software architecture, and probably not coincidentally, there has been a major uptake in the adoption of object technology. Object technology is a paradigm whose main benefits derive from a concentration on structure and a topological flexibility which allows it to be used to reflect more closely the structure of the problem space. Objects offer, above all, a different approach to software architecture because they liberate the development from the underlying digital architecture of the machine [O’Callaghan 1994, Cook 1994]. Within the community of object-oriented developers software patterns [Gamma 1995, Coplien 1995a, Fowler 1997] are being used to capture and reuse development expertise at various levels. This raises the intriguing possibility of the use of systems of software patterns, in the form of pattern languages, to help in the construction and reuse of software architectures.

There is strong circumstantial evidence for believing that software patterns can aid in the construction of software architectures. Design patterns document design approaches that are generally well-known (by experts) but rarely documented: Garlan and Shaw [Garlan 1996] similarly state that software architectures are well-known but are typically described in “idiosyncratic and informal” ways, if at all. Frank Buschmann [Buschmann 1996] and his colleagues at Siemens have gathered well-known architectural styles and developed them into architectural patterns in a system which also includes design patterns and
language-specific idioms (see Coplien 1992, Beck 1997 for C++ and Smalltalk idioms respectively). Finally, the Design Patterns book of Gamma et al., Gamma 1995, published with the subtitle "elements of reusable object-oriented software", was originally subtitled "micro-architectures for reuse".

2. Software Patterns and Pattern Languages

Patterns were introduced to the software industry by the already cited Gamma book, which was itself preceded by the work of Cunningham, Coplien, Coad, Anderson and others at successive OOPSLA workshops. Promoted by the Hillside Group, the patterns community has grown to become a worldwide network of software developers, practitioners and other professionals. Impressive though this growth has been, software patterns are immature when compared to, for example, the patterns of the built environment authored by Christopher Alexander and his colleagues Alexander 1977]. The most striking aspects of this relative immaturity are

- The disparity in the quality of published patterns
- The confinement of patterns, by and large, to a narrow field in the overall development of software
- The collection of patterns in catalogues rather than pattern languages

Alexander, whose work is a direct inspiration to the patterns movement, writes of his own domain that "...towns and buildings will not be able to come alive unless they are made by all the people in society and unless these people share a common pattern language within which to make these buildings, and unless this pattern language is alive itself. ...we present one possible pattern language... This language is extremely practical....

...The elements of this language are so-called patterns. Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [Alexander 1977, px]

3. A Pattern Language for Legacy System Migration

Architectural issues are never more to the fore than when an existing legacy system, constructed in other than object technologies, is being moved to objects creating, at least temporarily and sometimes permanently, a hybrid system. The move from, say, structured methods to object technology is a genuine paradigm shift in the Kuhnian sense (see Kuhn 1970, Cook 1994). It is possible not only to completely break away from the underlying architecture of the digital machine in developing an object-oriented software architecture, but also to develop a topology within which classes and objects capture the vocabulary of the problem domain. The range of possible application architectures is considerably widened. This overlays the fact that every legacy system itself has unique features, and specific issues caused by system entropy after years of maintenance. Developing and reusing guidelines for successful migrations requires finding meaningful abstractions that capture real success stories. Patterns enable us to do this. The author has now worked on four separate projects in which patterns have been extracted, identified and later reused in subsequent projects.

1 For example, the published CORBA patterns [Mowbray 1997] bear no relation to the patterns published in the proceedings of the Pattern Languages of Programming conferences [Coplien 1995, Vlissides 1996, Martin 1997]
Many of the most significant patterns already exist in the public domain as software patterns, others have analogues in existing catalogues. Still more are newly discovered patterns, specific to the migration activity. However, it is not only design patterns which have been discovered. In particular, organisation and process patterns [Coplien 1995b] have been found particularly useful in addressing the political issues which surround a migration effort, for example, which team “owns” a piece of the legacy architecture or code. What has been surprising is the way that patterns that spring from different sources interweave in an elegant way to generate outline solutions. It is this facet which has given rise to the idea that it may be possible to put together a critical mass of such patterns, with sufficient expressive power to be genuinely regarded as a pattern language, in the Alexandrian sense. We call this evolving language ADAPTOR which stands for Architecture-Driven and Pattern-based Techniques for Object Re-engineering. The language is architecture driven in the sense that it focuses on the crucial structural elements of moving a legacy system, and is therefore distinct from approaches which seek, for example, to reverse engineer legacy systems using formal methods to establish their functionality.

4 Thumbnails of ADAPTOR Patterns

As with other systems of patterns, the patterns in ADAPTOR are presented in the full form in a way dictated by a patterns template. The patterns themselves enter what is currently a catalogue after being peer reviewed in a pattern writers’ workshop [Rising 1997]. What is presented below are extremely brief thumbnails of some of the key, high-level patterns. This form has been chosen not only for brevity, but also to indicate as succinctly as possible the interconnectedness of the individual patterns.

**System Composite.** This pattern is an analogue of the Composite design pattern in the Gamma catalogue [Gamma 1995]. It is the most fundamental of the patterns – one which creates the context for the other patterns. System Composite views all systems as recursive aggregates being made up of subsystems of components and connectors. These subsystems can themselves be treated as systems in their own right. As with the Gamma Composite the power of the pattern is that these aggregates can be treated in the same way as system primitives. This permits system-level modelling techniques to be used at arbitrarily recursive depths in any large-scale system. **System Composite** is a pattern discovered by the Object Engineering and Migration group at De Montfort University.

**Scenarios Define the Problem.** This pattern exists in Jim Coplien’s generative software development pattern language [Coplien 1995b]. The pattern describes the utilisation of Use Cases or Task Scripts to capture interactions between external “actors” and the system to both capture functional requirements and drive the extraction of candidate classes etc. Actors can be humans, external systems or, applied recursively, other subsystems. The significance of using this pattern in the context set by System Composite is that it opens the way for Object-Oriented Analysis modelling of the problem space, including that part of it occupied by the legacy system. The use of similar techniques to forward engineering of object systems is fundamental to the approach described by ADAPTOR.

**Get the Model from the People.** This is a process pattern discovered in migration work done at BT by the Object Engineering and Migration group at De Montfort University. It was published at the first TelePlop (telecommunication Pattern Languages of Programs) workshop at OOPSLA ’96 [Wezeman 1996]. It focuses on the notion that a system maintainer’s hold in the heads and in their work culture valuable knowledge about the legacy system which is not held documented elsewhere. **Get the Model from the People** is actually itself the entry point into a small, self-contained pattern language.

**Three Layer Ingredients.** This pattern is a recent addition to ADAPTOR. It is based on Grady Booch’s multi-dimensional classification of software systems [Booch 1996]. For most of the archetypal systems he identifies, Booch notes a particular separation of concerns which is always tripartite, but which is unique to each different system category. He proposes “layering” the systems to explicitly factor out these different concerns. He does point out, however, that a large number of systems have the
characteristics of more than one of his categories and that more layering and partitioning is typically required (hence the notion of these being basic "ingredients" in a layered architecture). In conjunction with Scenarios define the Problem and Get the Model from the People this pattern can be used in early analysis to shape the high-level topology of the migrated system.

Facade. Façade is a Gamma pattern which describes an object which sits on the logical boundary between two subsystems. It presents a single interface of a subsystem to its clients, delegating requests for services of the subsystem to the actual objects (or other software entities in the case of a legacy system) which implement the requested behaviours. Façade is fundamental to the migration of legacy systems, allowing as it does, software on either side of the façade to evolve independently.

Semantic Wrapper. This pattern can be considered to be one that implements Facade in a legacy system context. The basic notion is that classes which exist to access legacy code should differ from other objects only in their implementation details. The interface they present to the rest of the system should capture and present abstract behaviours which have semantic content.

Detailed implementation strategies for interoperation between objects and legacy code are contained in linked patterns such as Narrow Interface and Service Access Layer (a.k.a. Thin Wrapper) which are both patterns, like Semantic Wrapper, discovered by the Object Engineering and Migration group. They form a small community of patterns with two other patterns, the Gamma Bridge pattern which promotes the construction of parallel "abstraction" and "implementation" hierarchies which allow incremental evolution of legacy implementations, and Remote Proxy (a.k.a. Ambassador [Coplien 1992]) which is a variant of the Gamma Proxy pattern, used in this context to represent legacies in distributed systems.

The set of patterns described above, together with others in the ADAPTOR catalogue can be used to establish the "scaffolding" for a migrated system. Work needs now to be allocated. Conway's Law, one of Coplien's organisational patterns, tells us that organisation must follow architecture or architecture must follow organisation. We choose the former, consciously moulding work organisation to the emerging software infrastructure. A first step in this direction can be taken with Code Ownership. By allocating responsibility for each Façade to a technical leader (one technical leader can have many such responsibilities, but no Façade should be without someone responsible for it) as the pattern suggests, work teams of developers will form around the key components identified earlier. Others of Coplien's patterns, such as Architect Also Implements and Developer Controls Process also have a role here, alongside a newly discovered pattern we call Keeper of the Flame which describes how continuity can be maintained with the architectural philosophy of the target system through subsequent iterations of the migration plan.

5 ADAPTOR – an Evolving Pattern Language

Successful migration work over a period of five years, including in projects prior to our attempts to “mine” patterns, gives us confidence in many of the fundamental patterns in the ADAPTOR catalogue. The individual patterns discovered by the Object Engineering and Migration group evolve themselves gaining greater expressiveness and power in each subsequent iterations, but we also have a number of candidate patterns whose nature has yet to be fully tested. It is anticipated that many candidate patterns will evolve further, but others will fail to survive the tests of experience. Moreover ADAPTOR is itself only a subset of the patterns needed to describe software architectures. It may prove to stand in the same relation to a wider language for developing software architecture that the mini-language, Get the Model From the People does to ADAPTOR itself. In particular we are interested in seeing whether Analysis Patterns [Fowler 1997] have a role to play in linking architectural solutions to models of the problem space itself. Certainly it seems to be the case that if a genuine, generative pattern language for OT migration is to
emerge it will combine organisational, process and design patterns from a number of wellsprings into a fluent combination. The work on ADAPTOR continues.
References


