Architecture Patterns

Introduction

Patterns for system architecting are very much in their infancy. They have been introduced into TOGAF essentially to draw them to the attention of the systems architecture community as an emerging important resource, and as a placeholder for hopefully more rigorous descriptions and references to more plentiful resources in future versions of TOGAF.

They have not (as yet) been integrated into TOGAF. However, in the following, we attempt to indicate the potential value to TOGAF, and to which parts of the TOGAF Architecture Development Method (ADM) they might be relevant.

Background

A "pattern" has been defined as: "an idea that has been useful in one practical context and will probably be useful in others" [Analysis Patterns - Reusable Object Models].

In TOGAF, patterns are considered to be a way of putting building blocks into context; for example, to describe a re-usable solution to a problem. Building blocks are what you use: patterns can tell you how you use them, when, why, and what trade-offs you have to make in doing so.

Patterns offer the promise of helping the architect to identify combinations of Architecture and/or Solution Building Blocks (ABBs/SBBs) that have been proven to deliver effective solutions in the past, and may provide the basis for effective solutions in the future.

Pattern techniques are generally acknowledged to have been established as a valuable architectural design technique by Christopher Alexander, a buildings architect, who described this approach in his book The Timeless Way of Building, published in 1979. This book provides an introduction to the ideas behind the use of patterns, and Alexander followed it with two further books (A Pattern Language and The Oregon Experiment) in which he expanded on his description of the features and benefits of a patterns approach to architecture.

Software and buildings architects have many similar issues to address, and so it was natural for software architects to take an interest in patterns as an architectural tool. Many papers and books have been published on them since Alexander's 1979 book, perhaps the most renowned being Design Patterns: Elements of Reusable Object-Oriented Software. This book describes simple and elegant solutions to specific problems in object-oriented software design.

Content of a Pattern

Several different formats are used in the literature for describing patterns, and no single format has achieved widespread acceptance. However, there is broad agreement on the types of things that a pattern should contain. The headings which follow are taken from Pattern-Oriented Software Architecture: A System of Patterns. The elements described below will be found in most patterns, even if different headings are used to describe them.

Name
A meaningful and memorable way to refer to the pattern, typically a single word or short phrase.

Problem
A description of the problem indicating the intent in applying the pattern - the intended goals and objectives to be reached within the context and forces described below (perhaps with some indication of their priorities).

Context
The preconditions under which the pattern is applicable - a description of the initial state before the pattern is applied.

Forces
A description of the relevant forces and constraints, and how they interact/conflict with each other and with the intended goals and objectives. The description should clarify the intricacies of the problem and make explicit the kinds of trade-offs that must be considered. (The need for such trade-offs is typically what makes the problem difficult, and generates the need for the pattern in the first place.) The notion of "forces" equates in many ways to the "qualities" that architects seek to optimize, and the concerns they seek to address, in designing architectures. For example:

- Security, robustness, reliability, fault-tolerance
- Manageability
- Efficiency, performance, throughput, bandwidth requirements, space utilization
- Scalability (incremental growth on-demand)
- Extensibility, evolvability, maintainability
- Modularity, independence, re-usability, openness, composability (plug-and-play), portability
- Completeness and correctness
- Ease-of-construction
- Ease-of-use
- etc., ...

Solution
A description, using text and/or graphics, of how to achieve the intended goals and objectives. The description should identify both the solution's static structure and its dynamic behavior - the people and computing actors, and their collaborations. The description may include guidelines for implementing the solution. Variants or specializations of the solution may also be described.

Resulting Context
The post-conditions after the pattern has been applied. Implementing the solution normally requires trade-offs among competing forces. This element describes which forces have been resolved and how, and which remain unresolved. It may also indicate other patterns that may be applicable in the new context. (A pattern may be one step in accomplishing some larger goal.) Any such other patterns will be described in detail under Related Patterns.

Examples
One or more sample applications of the pattern which illustrate each of the other elements: a specific problem, context, and set of forces; how the pattern is applied; and the resulting context.

Rationale
An explanation/justification of the pattern as a whole, or of individual components within it, indicating how the pattern actually works, and why - how it resolves the forces to achieve the desired goals and objectives, and why this is "good". The Solution element of a pattern describes the external structure and behavior of the solution: the Rationale provides insight into its internal workings.

Related Patterns
The relationships between this pattern and others. These may be predecessor patterns, whose resulting contexts correspond to the initial context of this one; or successor patterns, whose initial contexts correspond to the resulting context of this one; or alternative patterns, which describe a different solution to the same problem, but under different forces; or co-dependent patterns, which may/must be applied along with this pattern.

Known Uses
Known applications of the pattern within existing systems, verifying that the pattern does indeed describe a proven solution to a recurring problem. Known Uses can also serve as Examples.

Patterns may also begin with an Abstract providing an overview of the pattern and indicating the types of problems it addresses. The Abstract may also identify the target audience and what assumptions are made of the reader.

Terminology
Although design patterns have been the focus of widespread interest in the software industry for several years, particularly in the object-oriented and component-based software fields, it is only recently that there has been increasing interest in architecture patterns - extending the principles and concepts of design patterns to the architecture domain.

The technical literature relating to this field is complicated by the fact that many people in the software field use the term "architecture" to refer to software, and many patterns described as "architecture patterns" are high-level
software design patterns. This simply makes it all the more important to be precise in use of terminology.

**Architecture Patterns and Design Patterns**

The term "design pattern" is often used to refer to any pattern which addresses issues of software architecture, design, or programming implementation. In *Pattern-Oriented Software Architecture: A System of Patterns*, the authors define these three types of patterns as follows:

- **An Architecture Pattern** expresses a fundamental structural organization or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.
- **A Design Pattern** provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly recurring structure of communicating components that solves a general design problem within a particular context.
- **An Idiom** is a low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language.

These distinctions are useful, but it is important to note that architecture patterns in this context still refers solely to software architecture. Software architecture is certainly an important part of the focus of TOGAF, but it is not its only focus.

In this section we are concerned with patterns for enterprise system architecting. These are analogous to software architecture and design patterns, and borrow many of their concepts and terminology, but focus on providing re-usable models and methods specifically for the architecting of enterprise information systems - comprising software, hardware, networks, and people - as opposed to purely software systems.

**Patterns and the Architecture Continuum**

Although architecture patterns have not (as yet) been integrated into TOGAF, each of the first four main phases of the ADM (Phases A through D) gives an indication of the stage at which relevant re-usable architecture assets from the enterprise's Architecture Continuum should be considered for use. Architecture patterns are one such asset.

An enterprise that adopts a formal approach to use and re-use of architecture patterns will normally integrate their use into the enterprise's Architecture Continuum.

**Patterns and Views**

Architecture views are selected parts of one or more models representing a complete system architecture, focusing on those aspects that address the concerns of one or more stakeholders. Patterns can provide help in designing such models, and in composing views based on them.

**Patterns and Business Scenarios**

Relevant architecture patterns may well be identified in the work on business scenarios.

**Architecture Patterns in Use**

Two examples of architecture patterns in use are outlined in the following subsections, one from the domain of an IT customer enterprise's own architecture framework, and the other from a major system vendor who has done a lot of work in recent years in the field of architecture patterns.

- **The US Treasury Architecture Development Guidance (TADG) document** (see [US Treasury Architecture Development Guidance (TADG)](http://pubs.opengroup.org/architecture/togaf8-doc/arch/chap28.html)) provides a number of explicit architecture patterns, in addition to explaining a rationale, structure, and taxonomy for architectural patterns as they relate to the US Treasury.
- **The IBM Patterns for e-Business web site** (see [IBM Patterns for e-Business](http://pubs.opengroup.org/architecture/togaf8-doc/arch/chap28.html)) gives a series of architecture patterns that go from the business problem to specific solutions, firstly at a generic level and then in terms of specific IBM product solutions. A supporting resource is IBM's set of *Red Books*.

The following material is intended to give the reader pointers to some of the places where architecture patterns are already being used and made available, in order to help readers make their own minds up as to the usefulness of this technique for their own environments.

**US Treasury Architecture Development Guidance (TADG)**

The US Treasury Architecture Development Guidance (TADG) document - formerly known as the Treasury Information System Architecture Framework (TISAF) - provides a number of explicit architecture patterns.

Section 7 of the TADG document describes a rationale, structure, and taxonomy for architecture patterns, while the patterns themselves are formally documented in Appendix D. The architecture patterns presented embrace a larger set of systems than just object-oriented systems. Some architecture patterns are focused on legacy systems, some on concurrent and distributed systems, and some on real-time systems.

**TADG Pattern Content**

The content of an architecture pattern as defined in the TADG document contains the following elements:

- **Name**: Each architecture pattern has a unique, short descriptive name. The collection of architecture pattern names can be used as a vocabulary for describing, verifying, and validating Information Systems Architectures.

- **Problem**: Each architecture pattern contains a description of the problem to be solved. The problem statement may describe a class of problems or a specific problem.

- **Rationale**: The rationale describes and explains a typical specific problem that is representative of the broad class of problems to be solved by the architecture pattern. For a specific problem, it can provide additional details of the nature of the problem and the requirements for its resolution.

- **Assumptions**: The assumptions are conditions that must be satisfied in order for the architecture pattern to be usable in solving the problem. They include constraints on the solution and optional requirements that may make the solution more easy to use.

- **Structure**: The architecture pattern is described in diagrams and words in as much detail as is required to convey to the reader the components of the pattern and their responsibilities.

- **Interactions**: The important relationships and interactions among the components of the pattern are described and constraints on these relationships and interactions are identified.

- **Consequences**: The advantages and disadvantages of using this pattern are described, particularly in terms of other patterns (either required or excluded) as well as resource limitations that may arise from using it.

- **Implementation**: Additional implementation advice that can assist designers in customizing this architectural design pattern for the best results.

**TADG Architecture Patterns**

The TADG document contains the following patterns.

<table>
<thead>
<tr>
<th>Architectural Design</th>
<th>Pattern Name</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client-Proxy Server</td>
<td>Acts as a concentrator for many low-speed links to access a server.</td>
</tr>
<tr>
<td></td>
<td>Customer Support</td>
<td>Supports complex customer contact across multiple organizations.</td>
</tr>
<tr>
<td></td>
<td>Reactor</td>
<td>Decouples an event from its processing.</td>
</tr>
<tr>
<td></td>
<td>Replicated Servers</td>
<td>Replicates servers to reduce burden on central server.</td>
</tr>
<tr>
<td></td>
<td>Layered Architecture</td>
<td>A decomposition of services such that most interactions occur only between neighboring layers.</td>
</tr>
</tbody>
</table>
IBM Patterns for e-Business

The IBM Patterns for e-Business web site (www.ibm.com/framework/patterns) provides a group of re-usable assets aimed at speeding the process of developing e-Business applications. A supporting IBM web site is Patterns for e-Business Resources (www.ibm.com/developerworks/patterns/library). This is also known as the "Red Books".

The rationale for IBM's provision of these patterns is to:

- Provide a simple and consistent way to translate business priorities and requirements into technical solutions
- Assist and speed up the solution development and integration process by facilitating the assembly of a solution and minimizing custom one-of-a-kind implementations
- Capture the knowledge and best practices of experts and make it available for use by less experienced personnel
- Facilitate the re-use of intellectual capital such as reference architectures, frameworks, and other architecture assets

IBM's patterns are focused specifically on solutions for e-Business; i.e., those which allow an organization to leverage web technologies in order to re-engineer business processes, enhance communications, and lower organizational boundaries with:

- Customers and shareholders (across the Internet)
- Employees and stakeholders (across a corporate Intranet)
- Vendors, suppliers, and partners (across an Extranet)

They are intended to address the following challenges encountered in this type of environment:

- High degree of integration with legacy systems within the enterprise and with systems outside the enterprise
- The solutions need to reach users faster; this does not mean sacrificing quality, but it does mean coming up with better and faster ways to develop these solutions
- Service Level Agreements (SLAs) are critical
- Need to adapt to rapidly changing technologies and dramatically reduced product cycles
- Address an acute shortage of the key skills needed to develop quality solutions

IBM defines five types of pattern:

- **Business Patterns**, which identify the primary business actors, and describe the interactions between them in terms of different archetypal business interactions such as:
  - Service (a.k.a. user-to-business) - users accessing transactions on a 24x7 basis
  - Collaboration (a.k.a. user-to-user) - users working with one another to share data and information
  - Information Aggregation (a.k.a. user-to-data) - data from multiple sources aggregated and presented across multiple channels
  - Extended Enterprise (a.k.a. business-to-business) - integrating data and processes across enterprise boundaries

- **Integration Patterns**, which provide the "glue" to combine business patterns to form solutions. They characterize the business problem, business processes/rules, and existing environment to determine whether front-end or back-end integration is required.
  - Front-end integration (a.k.a. access integration) - focused on providing seamless and consistent access to business functions. Typical functions provided include single sign-on, personalization, transcoding, etc.
  - Back-end integration (a.k.a. application integration) - focused on connecting, interfacing, or integrating databases and systems. Typical integration can be based on function, type of integration, mode of integration, and by topology.

- **Composite Patterns**, which are previously identified combinations and selections of business and integration patterns, for previously identified situations such as: electronic commerce solutions, (public) enterprise portals, enterprise intranet portal, collaboration ASP, etc.

- **Application Patterns**. Each business and integration pattern can be implemented using one or more application patterns. An application pattern characterizes the coarse-grained structure of the application - the main application components, the allocation of processing functions and the interactions between them, the degree of integration between them, and the placement of the data relative to the applications.
**Runtime Patterns.** Application patterns can be implemented by run-time patterns, which demonstrate non-functional, service-level characteristics, such as performance, capacity, scalability, and availability. They identify key resource constraints and best practices.

The IBM web site also provides specific (IBM) product mappings for the run-time patterns, indicating specific technology choices for implementation.

### Some Pattern Resources

- The Patterns Home Page ([hillside.net/patterns](http://hillside.net/patterns)) hosted by the Hillside Group provides information about patterns, links to online patterns, papers, and books dealing with patterns, and patterns-related mailing lists.
- The Patterns-Discussion FAQ ([g.oswego.edu/dl/pd-FAQ/pd-FAQ.html](http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html)) maintained by Doug Lea provides a very thorough and highly readable FAQ about patterns.