BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Module 6: Contact Session 07

Patterns – Part 1 Suppli

Layers

Harvinder S Jabbal
SSZG653 Software Architectures

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Introducing Patterns

Patterns

An architectural pattern establishes a relationship between:

- ✓ *A context.*
- ✓ *A problem.*
- ✓ *A solution.*

Context

A context.

A recurring, common situation in the world that gives rise to a problem.

Problem

A problem.

The problem, appropriately generalized, that arises in the given context.

Solution

A solution.

A successful architectural resolution to the problem, appropriately abstracted. The solution for a pattern is determined and described by:

- A set of element types (for example, data repositories, processes, and objects)
- A set of interaction mechanisms or connectors (for example, method calls, events, or message bus)
- A topological layout of the components
- A set of semantic constraints covering topology, element behavior, and interaction mechanisms

Properties of Patterns

- Problem…Context…Solution
- Existing, well-proven design experience
- Identify and Specify Abstractions
- Provide Common Vocabulary
- Means of documenting Software Architecture
- Support construction with defined properties
- Help build complex and heterogenous architectures
- Help to manage software complexity

Problem…Context…Solution

- A pattern addresses a recurring design problem that arises in specific design situations, and presents a solution to it.

- If the problem is supporting variability in user interfaces. This problem may arise when developing software systems with human-computer interaction. You can solve this problem by a strict separation of responsibilities: the core functionality of the application is separated from its user interface.

Existing, well-proven design experience

- Patterns document existing, well-proven design experience.
- They are not invented or created artificially.
- Rather they 'distill and provide a means to reuse the design knowledge gained by experienced practitioners
- Those familiar with an adequate set of patterns 'can apply them immediately to design problems without having to rediscover them’
- Instead of knowledge existing only in the heads of a few experts, patterns make it more generally available.
- You can use such expert knowledge to design high-quality software for a specific task.
- The Model-View-Controller pattern, for example, presents experience gained over many years of developing interactive systems. Many well-known applications already apply the Model-View-Controller pattern-it is the classical architecture for many Smalltalk applications, and under several application frameworks such as MacApp/Windows Apps.

Identify and Specify Abstractions

- Patterns identify and specify abstractions that are above the level of single classes and instances, or of components.
- Typically, a pattern describes several components, classes or objects, and details their responsibilities and relationships, as well as their cooperation.
- All components together solve the problem that the pattern addresses, and usually more effectively than a single component.
- Example, the Model-View-Controller pattern describes a triad of three cooperating components, and each MVC triad also cooperates with other MVC triads of the system.

Provide Common Vocabulary

- Patterns provide a common vocabulary and understanding for design principles.
- Pattern names, if chosen carefully, become part of a widespread design language.
- They facilitate effective discussion of design problems and their solutions.
- They remove the need to explainn a solution to a particular problem with a lengthy and complicated description.
- Instead you can use a pattern name, and explain which parts of a solution correspond to which components of the pattern, or to which relationships between them.
- Example, the name 'Model-View-Controller' and the associated pattern has been to the Smalltalk community since the early '80s, and is used by many software engineers. When we say 'the architecture of the software follows Model-View-Controller', all our colleagues who are familiar with the pattern have an idea of the basic structure and properties of the application immediately.

Means of documenting Software Architecture

- Patterns are a means of documenting software architectures.
- They can describe the vision you have in mind when designing a software system.
- This helps others to avoid violating this vision when extending and modifying the original architecture, or when modifying the system's code.
- Example, if you know that a system is structured according to the Model-View-Controller pattern, you also know how to extend it with a new function: keep core functionality separate from user input and information display.

Support construction with defined properties

- Patterns support the construction of software with defined properties.
- Patterns provide a skeleton of functional behaviour and therefore help to implement the functionality of your application
- Example, patterns exist for maintaining consistency between cooperating components and for providing transparent peer-to-peer inter-process communication. In addition, patterns explicitly address non- functional requirements for software systems, such as changeability, reliability, testability or reusability.
- The Model-View-Controller pattern, for example, supports the changeability of user interfaces and the reusability of core functionality.

Help build complex and heterogenous architectures

- Patterns help you build complex and heterogeneous software architectures.
- Every pattern provides a predefined set of components, roles and relationships between them.
- It can be used to specify particular aspects of concrete software structures.
- Patterns 'act as building-blocks for constructing more complex designs'.
- This method of using predefined design artefacts supports the speed and the quality of your design.
- Understanding and applying well-written patterns saves time when compared to searching for solutions on your own.
- This is not to say that individual patterns will necessarily be better than your own solutions, but, at the very least. a pattern system can help you to evaluate and assess design alternatives.

….cont.

- However, although a pattern determines the basic structure of the solution to a particular design problem, it does not specify a fully detailed solution.
- A pattern provides a scheme for a generic solution to a family of problems, rather than a prefabricated module that can be used 'as is’.
- You must implement this scheme according to the specific needs of the design problem at hand.
- A pattern helps with the creation of similar units.
- These units can be alike in their broad structure, but are frequently quite different in their detailed appearance.
- Patterns help solve problems, but they do not provide complete solutions.

Help to manage software complexity

- Patterns help you to manage software complexity.
- Every pattern describes a proven way to handle the problem it addresses: the kinds of components needed, their roles, the details that should be hidden, the abstractions that should be visible, and how everything works.
- When you encounter a concrete design situation covered by a pattern there is no need to waste time inventing a new solution to your problem.
- If you implement the pattern correctly, you can rely on the solution it provides.
- The Model-View-Controller pattern, for example, helps you to separate the different user interface aspects of a software system and provide appropriate abstractions for them.

Definition: Pattern

A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate.

Categories

From Mud to Structure.

- Patterns in this category help you to avoid a 'sea' of components or objects.
- In particular, they support a controlled decomposition of an overall system task into cooperating subtasks.
- The category includes
  - the Layers pattern
  - the Pipes and Filters pattern
  - the Blackboard pattern

Distributed Systems.

- This category includes one pattern.
  - Broker
- and refers to two patterns in other categories,
  - Microkernel
  - Pipes and Filters
- The Broker pattern provides a complete infrastructure for distributed applications.
- The Microkernel and Pipes and Filters patterns only consider distribution as a secondary concern and are therefore listed under their respective primary categories.

Interactive Systems.

- This category comprises two patterns,
  - the Model-View-Controller pattern (well-known from Smalltalk,)
  - the Presentation-Abstraction-Control pattern.
- Both patterns support the structuring of software systems that feature human-computer interaction.

Adaptable Systems.

- This category includes
  - The Reflection pattern
  - the Microkernel pattern
- strongly support extension of applications and their adaptation to evolving technology and changing functional requirements.

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

From Mud to Structure: Layers

From Mud to Structure: Layers

Context:

All complex systems experience the need to develop and evolve portions of the system independently. For this reason the developers of the system need a clear and well-documented separation of concerns, so that modules of the system may be independently developed and maintained.

From Mud to Structure: Layers

Problem:

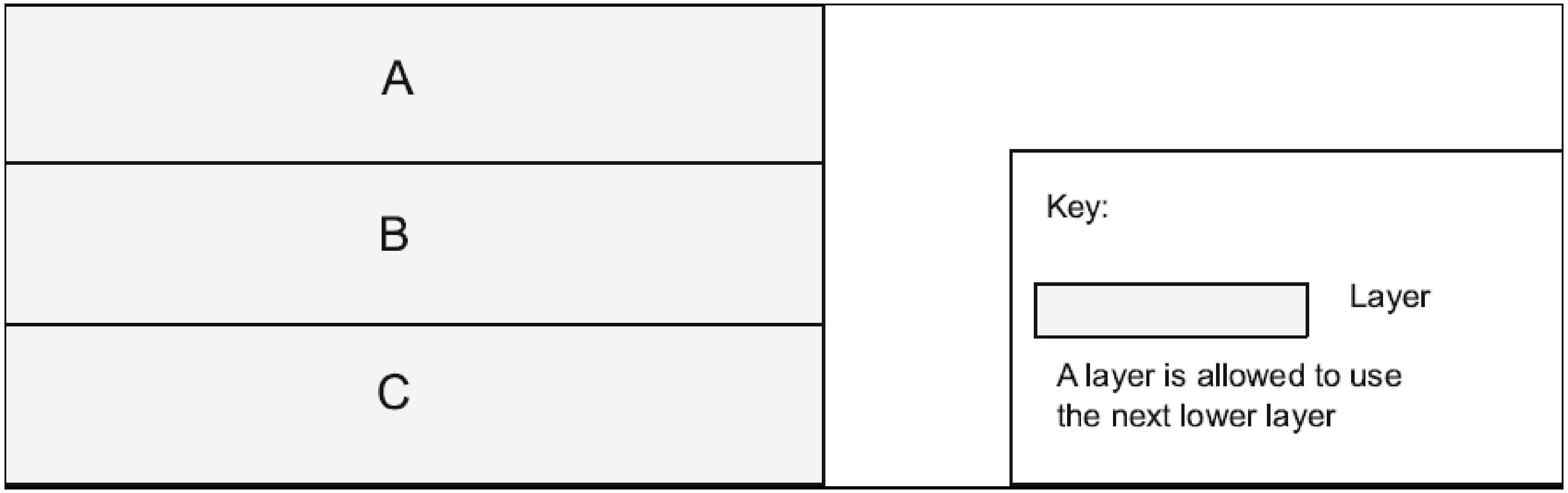
The software needs to be segmented in such a way that the modules can be developed and evolved separately with little interaction among the parts, supporting portability, modifiability, and reuse.

From Mud to Structure: Layers

Solution:

To achieve this separation of concerns, the layered pattern divides the software into units called layers. Each layer is a grouping of modules that offers a cohesive set of services. The usage must be unidirectional. Layers completely partition a set of software, and each partition is exposed through a public interface.

Layer: Diagramatical Representation



A Typical Solution

Overview:

The layered pattern defines layers (groupings of modules that offer a cohesive set of services) and a unidirectional *allowed-to-use* relation among the layers.

Elements:

Layer, a kind of module. The description of a layer should define what modules the layer contains.

Relations:

Allowed to use. The design should define what the layer usage rules are and any allowable exceptions.

Constraints:

- Every piece of software is allocated to exactly one layer.
- There are at least two layers (but usually there are three or more).
- The *allowed-to-use* relations should not be circular (i.e., a lower layer cannot use a layer above).

Weaknesses:

- The addition of layers adds up-front cost and complexity to a system.
- Layers contribute a performance penalty.