Architektur Patterns

Vorbemerkung:

Es wurde darauf verzichtet englische Worte wie Pattern, Layer, Black-box usw. ins Deutsche zu übersetzten, da in diesen Fällen die englischen Worte bezeichnender sind.

Was sind Architektur Pattern? 1/3

1. Allgemeine Definition:

Architektur Pattern beschreiben fundamentale strukturelle Organisationsschemata für Software Systeme. Sie bieten eine Zusammenstellung von vordefinierten Subsystemen, bestimmen ihre Verantwortlichkeiten und beinhalten Regeln und Leitlinien für die Organisation der Beziehungen zwischen diesen Systemen.

Was sind Architektur Pattern? 2/3

2. Software–Architektur:

Eine Software–Architektur

- ist eine Beschreibung von
  - Subsystemen
  - Komponenten
  - Beziehungen zwischen diesen
- ist ein Artefakt
- entsteht durch Aktivitäten des Systementwurfs

Was sind Architektur Pattern? 3/3

3. Architekturpattern:

Ein Architekturpattern

- beschreibt die grundlegende Organisationsstruktur von Software–Systemen
- beschreibt systemweite Eigenschaften
- ist eine Schablone für konkrete Software–Architekturen
- ist eine Abstraktion konkreter Software–Architekturen

Deshalb ist die Wahl eines Architekturpattern eine weitreichende Entscheidung bei der Entwicklung von Software–Systemen.

Frank Buschmann teilt die Architektur Pattern in vier Kategorien ein: 1/5

1. From Mud to Structure. (Vom Schlamm zur Struktur)

- Ziel ist das Vermeiden eines „Sees" von Komponenten oder Objekten.
- Erreicht wird dies durch Zergliedern des ganzen Systems auf hohem Level.
- Beispiele sind das Layer Pattern, das Pipes and Filters Pattern und das Blackboard Pattern.

Frank Buschmann teilt die Architektur Pattern in vier Kategorien ein: 2/5

2. Distributed Systems (Verteilte Systeme)

- Diese Kategorie umfasst ein Pattern (Broker Pattern) und verweist auf zwei andere Pattern in anderen Kategorien (Microkernel und Pipes and Filters).
- Das Broker Pattern unterstützt beispielsweise eine komplette Infrastruktur für verteilte Anwendungen. Seine Architektur ist schon von der Object Management Group (OMG) standardisiert.

Frank Buschmann teilt die Architektur Pattern in vier Kategorien ein: 3/5

3. Interactive Systems (Interaktive Systeme)

- Diese Kategorie enthält zwei Pattern: das Model-View-Controller Pattern (bekannt von Smalltalk) und das Presentation-Abstraction-Control Pattern.
- Beide Pattern unterstützen die Strukturierung von Softwaresystemen dessen Besonderheit die Mensch-Computer Interaktion ist.

Frank Buschmann teilt die Architektur Pattern in vier Kategorien ein: 4/5

4. Adaptable Systems (anpassbare Systeme)

- Das Reflection Pattern und das Microkernelpattern unterstützt die Erweiterung von Applikationen und deren Anpassung an sich entwickelnde Technologien und sich ändernde funktionale Anforderungen.

Frank Buschmann teilt die Architektur Pattern in vier Kategorien ein: 5/5

Näheres zu „1. From Mud to Structure"

- Optimistisch wird angenommen, dass die Anforderungen an unser neues System wohl überlegt und stabil sind.
- Ziel ist es, den unüberschaubaren „ball of mud" in eine arbeitende Struktur zu bringen, wobei alle Aspekte der Anforderungen zu berücksichtigen sind.
- Dazu wird eine Zergliederung des ganzen Systems auf hohen Level vorgenommen, wie beispielsweise eine horizontale Zergliederung in „Hierarchische Schichten" (Layer Pattern) oder eine vertikale Zergliederung (Pipes and Filters).

Layer Architectural Pattern

Was ist das Layer Pattern 1/3

Allgemeine Definition:

Das **Layer (Schichten)** Pattern hilft Applikationen zu strukturieren, die in Gruppen von Unteraufgaben (Subtasks) zerlegt werden können. Jede dieser Gruppen befindet sich in einer besonderen Level der Abstraktion.

Was ist das Layer Pattern 2/3

Kontext: Große Systeme, die einer Zerlegung bedürfen.

Problem: Große Software–Systeme besitzen häufig eine Mischung aus Operationen mit unterschiedlichen Abstraktionsgraden.

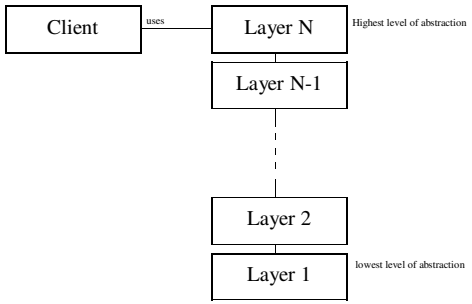
- Operationen höheren Abstraktionsgrades beziehen sich auf niedrigere.
- Code-Änderungen sollen sich auf die jeweilige Komponente beschränken (Lokalität).
- Robustheit der Schnittstellen
- Austauschbarkeit von Komponenten, Alternativimplementierungen
- Wiederverwendbarkeit von Komponenten

Was ist das Layer Pattern 3/3

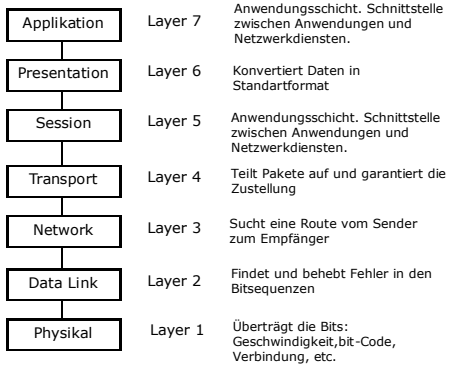
Lösung:

- Gruppiere Operationen gleichen Abstraktionsgrades zusammen.
- Dabei entsteht für jeden Grad der Abstraktion ein **Layer**.
- Dienste höherer Schichten stützen sich auf Dienste niedrigerer Schichten ab.
- Layer möglichst weit voneinander entkoppeln (Lokalität): Layer interagieren jeweils nur mit nächst höherem und nächst unterem Layer.
- Eine einheitliche Schnittstelle für jedes Layer erhöht die Austauschbarkeit, Wiederverwendbarkeit.

Schematische Darstellung der Layer



Netzwerkprotokolle sind wahrscheinlich das best-bekannteste Beispiel von strukturierter Architektur.



Ein individuelle Schicht kann durch folgende CRC Karte beschrieben werden:

Class Layer J	**Collaborator** • Layer J-1
Responsibility • Provides services used by Layer J+1. • Delegates subtasks to Layer J-1.	

Dynamisches Verhalten von „layered applications" 1/5

Die folgenden vier Szenarien sind Archetypen für das dynamische Verhalten von „layered applications"

Das erste Szenario ist wahrscheinlich das Bekannteste. Ein Anwender sendet einen Request an Layer N. Layer N kann diesen Request nicht alleine ausführen und ruft unterstützende Subtasks von Layer N-1. Layer N-1 bearbeitet diesen Request und sendet weitere Requests an Layer N-2 und so weiter bis Layer 1 erreicht ist. Hier sind die „lowest-level services" schließlich implementiert. Wenn es notwendig ist, werden die Antworten auf die verschiedenen Anfragen von Layer 1 zu Layer 2 und von Layer 2 zu Layer 3, und so weiter, durchgegeben, bis die letzte Antwort Layer N erreicht hat.

Dynamisches Verhalten von „layered applications" 2/5

Das zweite Szenario illustriert eine „bottom-up communication" - eine Kette von Aktionen, die in Layer 1 starten, z.B. wenn ein Device-Treiber einen Input bekommt. Der Treiber übersetzt diesen Input in ein internes Format und sendet diesen an Layer 2, welches anfängt, den Input zu interpretieren und so weiter. Auf diesem Weg wandern die Daten durch die verschiedenen Layer bis das höchste Layer erreicht ist.
„Top-down" Informationen werden als Request und „bottom-up calls" als Notifications bezeichnet.

Dynamisches Verhalten von „layered applications" 3/5

Das dritte Szenario beschreibt die Situation, wenn Requests nur durch einen Teil der Layer laufen. Ein „top-level" Request geht z.B. nur zu Layer N-1, wenn dieses Layer den Request erfüllen kann.
Dies ist z.B. der Fall, wenn Layer N-1 als Cache agiert. So kann der Request von Layer N erfüllt werden, ohne durch alle Schichten zu Layer 1 zu laufen und von dort aus zu einem Remote-Server.

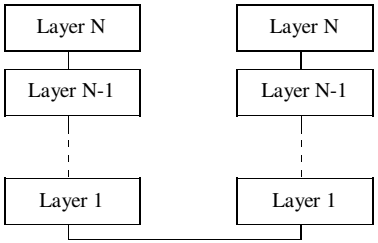
Dynamisches Verhalten von „layered applications“ 4/5

Das vierte Szenario beschreibt eine dem Szenario 3 sehr ähnlichen Situation. Ein Ereignis wird in Layer 1 entdeckt, aber stoppt in Layer 3 und wandert nicht bis zu Layer N weiter. In einem Kommunikationsprotokoll könnte beispielsweise ein erneuter Request von einem ungeduldigen Client ankommen, der kurze Zeit vorher schon Daten angefordert hat. Jedoch hat der Server in der Zwischenzeit bereits diese Daten gesendet. Somit überschneidet sich die Antwort und die nochmalige Datenanfrage. In diesem Fall bemerkt dies Layer 3 und bearbeitet die nochmalige Datenanfrage ohne weitere Aktionen.

Dynamisches Verhalten von „layered applications“ 5/5

Das fünfte Szenario beschreibt, wie zwei Layer miteinander kommunizieren. Dies ist gut von Netzwerkprotokollen bekannt. Im folgenden Diagramm schickt Layer N einen Request. Dieser Request geht durch alle Schichten hindurch, bis er schließlich Layer 1 erreicht hat, wird dann zu Layer 1 vom rechten „Stack“ weitergeleitet und von dort aus wieder nach oben bis Layer N. Die Antwort geht genauso rückwärts, bis sie Layer N des linken „Stack“ erreicht hat.

Schematische Darstellung des fünften Szenarios:



Implementierung einer „layered application“ 1/8

Die Implementierung gliedert sich in folgende Schritte. Obwohl nicht alle dieser Schritte in jeder Applikation notwendig oder sinnvoll sind.

1. Definiere das Abstraktionskriterium um die Tasks (Aufgaben) in Layern zu gruppieren.

In der Softwareenwicklung benutzen wir oft einen Mix von abstraktions Kriterien. Zum Beispiel kann der Abstand von der Hardware die unteren Layer bilden und die begrifflich komplexeren formen die oberen Layer.

Dies kann z.B. so aussehen (von oben nach unten geordnet):

- für den Benutzer sichtbare Elemente
- konkrete Applikationsmodule
- allgemeine Aufgabensebene
- Betriebssystem Interface
- Betriebssystem
- Hardware

Implementierung einer „layered application“ 2/8

2. Bestimme die Anzahl der Abstraktionslevel entsprechend den Abstraktionskriterien.

Jedes Abstraktionslevel entspricht einem Layer des Patterns.

Manchmal ist dies aber nicht sinnvoll. Ergibt es zu vielen Layer wird unnötiger overhead produziert und zu wenig Layer können in einer armen Struktur enden.

3. Benenne die Layer und weise ihnen Aufgaben zu.

Die Aufgabe des höchsten Layer ist der allumfassende Systemtask wie er vom Client wahrgenommen wird. Die Aufgaben der anderen Schichten ist es, der oberen zu helfen.

Implementierung einer „layered application“ 3/8

4. Präzisiere die Dienste.

Das wichtigste Implementationsprinzip ist es, dass die Schichten strikt getrennt voneinander sind. So wird verhindert, dass eine Komponente sich über mehr als ein Layer ausbreitet.

Es ist oft besser in den höheren Schichten mehr Dienste als in den unteren Schichten zu implementieren. Das hat den Vorteil, dass Entwickler sich nicht mit vielen „low-level primitives“ befassen müssen, welche sich während des Entwicklungsprozesses noch ändern könnten. So werden die Basislayer dünn gehalten, während die oberen Layer ein breiteres Spektrum von Einsatzmöglichkeiten besitzen. Dieses Phänomen wird auch als „inverted pyramid of reuse“ genannt.

Implementierung einer „layered application“ 4/8

5. Verfeinere das layering.

Wiederhole Schritt 1-4. Es ist normalerweise nicht möglich genaue Abstraktionskriterien zu definieren, bevor man über die Layer und ihre Dienste nachgedacht hat.
Alternativ ist es normalerweise falsch, erst die Komponenten und Dienste zu definieren und später eine Schichtenstruktur entsprechend ihrer Beziehungen zu einander darüber zu legen. Dadurch kann es leicht passieren, dass die Systemerhaltung die Architektur zerstört. Z.B. könnte eine neue Komponente Anfragen an mehr als nur eine Schicht senden und damit das Prinzip der strikten Trennung der Layer verletzten.

Die Lösung ist es, die ersten vier Schritte so oft aus zu führen bis sich ein natürliches und stabiles Layering ergibt.

Implementierung einer „layered application“ 5/8

6. Bestimme ein Interface für jedes Layer.

Wenn Layer J eine „black box“ für Layer J+1 ist muss ein Interface geschrieben werden, welches alle Dienste von Layer J anbietet.
Eine „white box“ Lösung wäre es, wenn Layer J+1 ins „Innere“ von Layer J sehen könnte.
„gray box“ stellt einen Kompromiss aus den beiden anderen dar. Hier weiß Layer J+1 von drei Komponenten aus Layer J und kann diese separat ansprechen, weiß jedoch nichts von der inneren Arbeitsweise von individuellen Komponenten.

Es ist gute Designpraxis die „Black-Box“ zu verwenden, wenn immer es möglich ist. Es unterstützt die Systementwicklung besser als andere Ansätze.

Implementierung einer „layered application“ 6/8

7. Strukturiere die individuellen Layer.

Traditionell lag der Fokus auf der geeigneten Beziehung zwischen den Layer, aber im inneren herrschte ein freilaufendes Chaos. Wenn in individuelles Layer komplex ist, sollte es in separate Komponenten aufgeteilt werden. Diese Unterteilung kann durch feinkörnige Patterns unterstützt werden. Z.B. das Brigde oder Strategy Pattern.

8. Bestimme die Kommunikation zwischen benachbarten Layer.

Der meist genutzte Mechanismus für die Layer-Layer Kommunikation ist das „Push-Model“. Wenn Layer J einen Dienst von Layer J-1 beansprucht wird jede benötigte Information als Teil des Service-Call übergeben. Der umgekehrte Fall ist als „pull-model“ bekannt und kommt vor, wenn das untere Layer sich die verfügbaren Informationen vom höheren Layer selber besorgt. Das „Publisher-Subscriber“ und „Pipes and Filters“ Pattern liefern weitere Informationen über das „push und pull-Model“.

Implementierung einer „layered application“ 7/8

9. Entkopple benachbarte Layer.

Um dies um zu setzten gibt es viele Möglichkeiten.
Oft weiß ein höheres Layer vom nächst unteren Layer, aber das untere Layer ist in Unkenntnis über seinen User. Hier besteht nur eine „one-way“ Kopplung. Veränderungen in Layer J müssen so das Layer J+1 nicht berücksichtigen. Solch eine Kopplung ist perfekt, wenn Requests top-down laufen.

Für eine bottom-up Kommunikation können Callbacks verwendet werden und immernoch eine top-down one-way Kopplung erhalten bleiben. Hier implementiert das obere Layer Callbackfunktionen für das untere Layer. Das ist vorallem sehr effektiv, wenn nur eine feste Anzahl von möglichen Events vom unterem zum höheren Layer gesendet werden. Während dem Start-up teilt das höhere Layer dem tieferem Layer mit, welche Funktionen gerufen werden sollen, falls ein bestimmtes Event eintritt. Das untere Layer merkt sich diese Funktionen in einem Register. Ein Beispiel wäre das Reactor-Pattern.

Implementierung einer „layered application“ 8/8

10. Entwerfe eine Error-Handling Strategie

Error-Handling kann für Layer-Architekturen teuer in Bezug auf Bearbeitungszeit und dem Programmieraufwand sein. Ein Error kann entweder in dem Layer behandelt werden, indem er auftritt oder zum nächst höheren Layer weitergebenen werden. Im ersten Fall muss das niedrigere Layer den Error in eine Errorbeschreibung umwandeln, so dass es das höhere Layer versteht. Eine Daumenregel besagt, die Errors möglichst auf dem niedrigst möglichen Layer ab zu fangen. Das bewahrt die höheren Layer davor von vielen unterschiedlichen Errors überflutetet zu werden und vor umfangreichen Error-Handling-Code.
Versucht werden sollte zumindest, ähnliche Error-Typen zu einem allgemeineren Error-Typ (general error types) zusammen zu fassen. Weiterverbreitet werden dann nur diese general error types. Wird dies nicht gemacht, könnte es passieren, dass höhere Layer mit Errors konfrontiert werden, die eigentlich von niedrigeren Layern behandelt werden sollten. Diese Errors werden dann von den höheren Layern nicht verstanden und wer kennt nicht die total kryptischen Error Nachrichten, die bis zum Userlayer durchgeben wurden und keiner versteht?

Literaturverweise zum Vortrag:

- F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: A System of Patterns, Wiley & Sons. 1996
- Dr. Andreas Harrer, Muster in der Software–Technik, Vortrag 6. Mai 2004