Universität Zürich[UZH]

Institut für Informatik

RERG

Architekturmuster und Entwurfsmuster

1. Architekturmuster

1.1 Chaos zu Struktur (engl. Mud-to-structure)

Diese Muster sollen helfen die Vielzahl der Komponenten und Objekte eines Softwaresystems zu organisieren. Die Funktionalität des Gesamtsystems wird hierbei in kooperierende Subsysteme aufgeteilt.

Pipes und Filter: Beschreibt die Struktur für Systeme, die Datenströme verarbeiten. Das System wird durch mehrere unabhängige Einheiten strukturiert: Verarbeitungsschritte, sogenannte Filter mit einer Umwandlung (beispielsweise Veränderung, Ergänzung, Entfernung) von Daten, und Verbindungen zwischen den Filtern, sogenannte Pipes zur effizienten Weiterleitung der Daten.

Schwarzes Brett bzw. Blackboard: Ein Architekturmuster zur Bewältigung von Problemlöseprozessen. Auf dem Blackboard werden dabei von einzelnen Teilprozessen Daten in einer hierarchisch organisierten Form abgelegt. Das Blackboard ist nun in der Lage, andere Teilprozesse von der Ablage oder Änderung dieser Daten zu benachrichtigen. Dies ermöglicht eine nahezu parallele Arbeitsweise der Teilprozesse.

Schichtenarchitektur: Trennt die Bestandteile eines Softwaresystems in aufeinander aufbauende Schichten

Domain-Driven Design und Naked Objects: Anwendungsdomänen-getriebene Herangehensweise an Architektur und Design

Data Context Interaction: Trennung von Fachobjekten (Data) Use-Cases und Algorithmen (Context) und fachlichen Rollen (Interaction).

Command Query Responsibility Segregation: Trennung des Businessmodells in Teile, die Daten holen (Query) und Teile, die Daten ändern bzw. Fachlichkeiten ausführen (Command)

1.2 Verteilte Systeme

Diese Kategorie unterstützt die Verwendung verteilter Ressourcen und Dienste in Netzwerken (z. B. serviceorientierte Architekturen, Orchestrierung). Zwei weitere Modelle ('Mikrokernel' und 'Pipes und Filter') unterstützen Verteilung zweitrangig.

Serviceorientierte Architektur (SOA): Ein Architekturmuster für verteilte Systeme, um Dienste von IT-Systemen zu strukturieren und zu nutzen. Durch Zusammensetzen (Orchestrierung) werden damit Geschäftsprozessen durch Serviceimplementierungen abgebildet. Durch Orchestrierung von Services niedriger Abstraktionsebenen können so recht flexibel und unter Ermöglichung größtmöglicher Wiederverwendbarkeit Services höherer Abstraktionsebenen geschaffen werden.

Peer-to-Peer: Bezeichnet eine Möglichkeit Aufgaben und Dienstleistungen innerhalb eines Netzwerkes zu verteilen. Dabei sind alle Computer gleichberechtigt und können sowohl Dienste in Anspruch nehmen, als auch zur Verfügung stellen. Ebenso können die Netzwerkteilnehmer abhängig von ihrer Qualifikation in verschiedene Gruppen eingeteilt werden, und spezifische Aufgaben

übernehmen.

Client-Server: Beschreibt eine weitere Möglichkeit, Aufgaben und Dienstleistungen innerhalb eines Netzwerkes zu verteilen. Die Aufgaben werden von Programmen erledigt, die in Clients und Server unterteilt werden. Der Client kann auf Wunsch eine Aufgabe vom Server anfordern, der Server beantwortet die Anforderung.

1.3 Interaktive Systeme

Muster dieser Kategorie helfen, Mensch-Computer-Interaktionen zu strukturieren.

Model View Controller (MVC) und Model View Presenter: Model View Controller teilen die Benutzerinterface-Interaktionen in drei verschiedene Rollen. Das Modell enthält die darzustellenden Daten und die Geschäftslogik. Es ist von Präsentation und Steuerung unabhängig. Die View ist für die Darstellung der benötigten Daten aus dem Modell und die Entgegennahme von Benutzerinteraktionen zuständig. Sie kennt sowohl ihre Steuerung (Controller) als auch das Modell, dessen Daten sie präsentiert, ist aber nicht für die Weiterverarbeitung der vom Benutzer übergebenen Daten zuständig. Die Steuerung (Controller) nimmt von der View Benutzeraktionen entgegen, wertet diese aus und agiert entsprechend.

Presentation-Abstraction-Control (PAC): Ein Architekturmuster zur Strukturierung von interaktiven Softwaresystemen. Dabei werden diese derart in Teile zerlegt, dass jeder Teil genau eine Aufgabe des Systems anbietet. Damit wird eine hohe Flexibilität des Systems erhalten, und man muss sich nur darum kümmern, dass diese Teile zu einem funktionierenden Ganzen zusammengesetzt werden und auch zusammenarbeiten.

2. Entwurfsmuster

2.1 Erzeugungsmuster (Creational Patterns)

Abstract factory: Es definiert eine Schnittstelle zur Erzeugung einer Familie von Objekten, wobei die konkreten Klassen der zu instanziierenden Objekte nicht näher festgelegt werden.

Factory method: Das Muster beschreibt, wie ein Objekt durch Aufruf einer Methode anstatt durch direkten Aufruf eines Konstruktors erzeugt wird.

Builder: Es trennt die Konstruktion komplexer Objekte von deren Repräsentationen, wodurch dieselben Konstruktionsprozesse wiederverwendet werden können.

Prototype: Neue Instanzen werden auf Grundlage von prototypischen Instanzen („Vorlagen“) erzeugt. Dabei wird die Vorlage kopiert und an neue Bedürfnisse angepass.

Singleton: Es stellt sicher, dass von einer Klasse genau ein Objekt existiert. Dieses Singleton ist darüber hinaus üblicherweise global verfügbar

2.2 Strukturmuster (Structural Patterns)

Adapter: Das Muster dient zur Übersetzung einer Schnittstelle in eine andere. Dadurch wird die Kommunikation von Klassen mit zueinander inkompatiblen Schnittstellen ermöglicht.

Bridge: dient zur Trennung der Implementierung von ihrer Abstraktion (Schnittstelle), wodurch beide unabhängig voneinander verändert werden können.

Composite: Das Kompositionsmuster (*composite pattern*) wird angewendet, um Teil-Ganzes-Hierarchien zu repräsentieren, indem Objekte zu Baumstrukturen zusammengefügt werden. Die Grundidee des Kompositionsmusters ist, in einer abstrakten Klasse sowohl primitive Objekte als auch ihre Behälter zu repräsentieren. Somit können sowohl einzelne Objekte, als auch ihre Kompositionen

einheitlich behandelt werden.

Decorator: Das Muster ist eine flexible Alternative zur Unterklassenbildung, um eine Klasse um zusätzliche Funktionalitäten zu erweitern.

Façade: Es bietet eine einheitliche und meist vereinfachte Schnittstelle zu einer Menge von Schnittstellen eines Subsystems

Flyweight: Das Fliegengewicht wird verwendet, wenn eine große Anzahl von Objekten benötigt wird, die sich bestimmte variable Informationen teilen und eine herkömmliche Implementierung unverhältnismäßig viele Ressourcen erfordern

Proxy: Das Muster dient zum Verschieben der Kontrolle über ein Objekt auf ein vorgelagertes Stellvertreterobjekt.

2.3 Verhaltensmuster (Behavioral Patterns)

Chain of responsibility: Dabei dient es der Entkopplung des Auslösers einer Anfrage mit seinem Empfänger.

Command: In diesem Entwurfsmuster kapselt das Kommando-Objekt einen Befehl, um es so zu ermöglichen, Operationen in eine Warteschlange zu stellen, Logbucheinträge zu führen und Operationen rückgängig zu machen

Interpreter: Das Interpretermuster definiert eine Repräsentation für die Grammatik einer Sprache und die Möglichkeit, Sätze dieser Sprache zu interpretieren.

Iterator: Es stellt Möglichkeiten zur Verfügung, auf Elemente einer aggregierten Struktur sequenziell zuzugreifen, ohne die Struktur zu enthüllen.

Mediator: Das Muster dient zum Steuern des kooperativen Verhaltens von Objekten, wobei Objekte nicht direkt kooperieren, sondern über einen Vermittler.

Memento: Das Muster dient zur Erfassung und Externalisierung des internen Zustands eines Objektes, wobei sichergestellt wird, dass dadurch seine Kapselung nicht verletzt wird. So kann das Objekt zu einem späteren Zeitpunkt wieder in diesen Zustand zurückversetzt werden

Observer: Dient der Weitergabe von Änderungen an einem Objekt an von diesem Objekt abhängige Strukturen.

State: Das Zustandsmuster wird zur Kapselung unterschiedlicher, zustandsabhängiger Verhaltensweisen eines Objektes eingesetzt.

Strategy: Das Zustandsmuster wird zur Kapselung unterschiedlicher, zustandsabhängiger Verhaltensweisen eines Objektes eingesetzt.
Die Algorithme koenen werend der Laufzeit ausgetauscht werden.

Template: Teilschritte eines Algorithmus variabel gehalten werden können

Visitor: Es dient der Kapselung von Operationen, die es ausgeführt auf Elementen einer Objektstruktur ermöglichen, dass neue Operationen ohne Veränderung der betroffenen Elementklassen definiert werden.

Literatur

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: elements of reusable objectoriented software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.