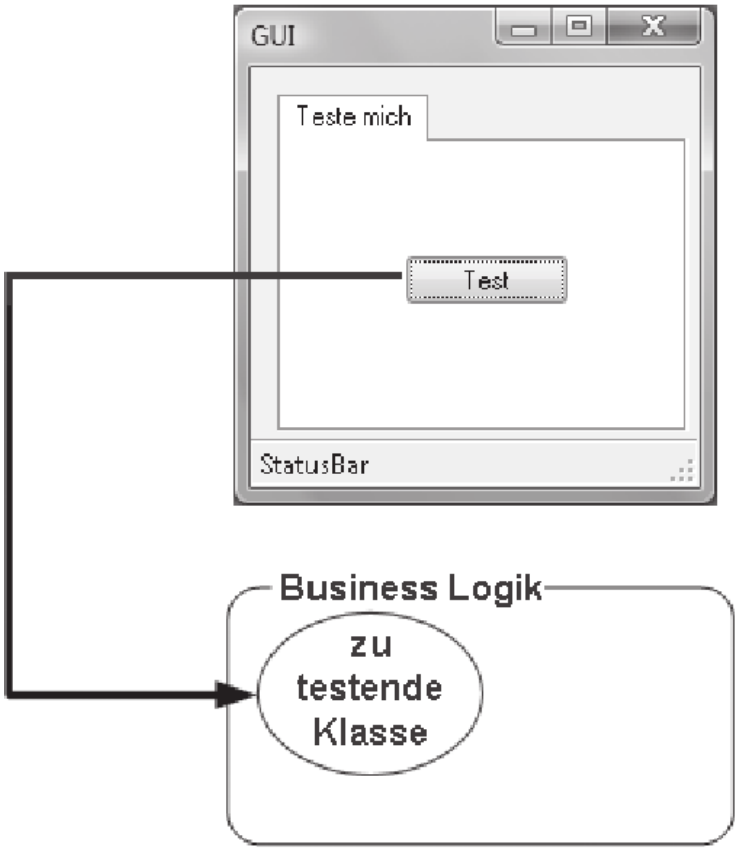
Zusammenfassung - Softwarekomponenten

19 February 2015 13:17

Version: 1.0.0
Study: 2. Semester, Bachelor in Business and Computer Science
School: Hochschule Luzern - Wirtschaft
Author: Janik von Rotz (http://janikvonrotz.ch)

Unit Testing

23 February 2015 13:31



- Ein Unit Test ist ein Stück Code, das ein andere Stück Code aufruft und anschliessend die Richtigkeit einer oder mehreren Prüfungen bestimmt.
- Sind die Ergebnisse falsch, gilt dies als Unit Test fehlschlag.
- Eine Unit ist eine Methode oder Funktion
- In einem Softwareprojekt werden für alle Komponente sprich tausende solche Unit Tests durchgeführt.

Gute Unit Test

- Da der Zeit-Aufand realtiv gross ist um Unit Tests zu schreiben ergeben sich in der Realität zwangsweise Probleme in der Umsetzung und Entwicklung von Softwarekompontenten.
- Gerade unter Einhaltung von Zeitplänen kommen Unit Tests zu kurz.
- Diese Reaktion hat im weiteren Verlauf von Projekten einen massgebenden Einfluss auf die Qualität des Codes.
- Es muss klar definiert werden was einen guten und vor allem brauchbaren Unit Test ausmacht.

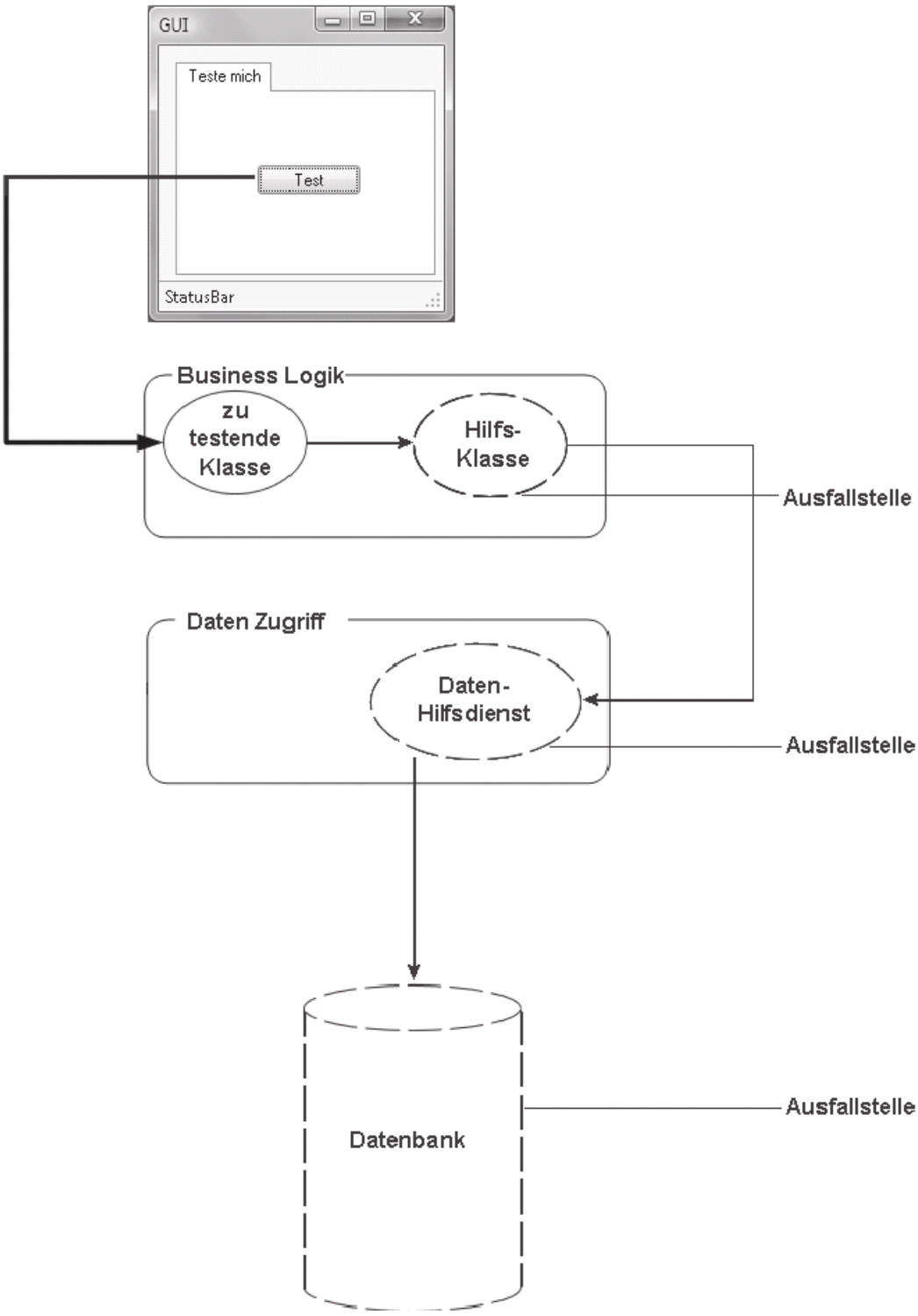
Eigenschaften eines guten Unit Tests

- Automatisiert und wiederholbar
- Einfach zu implementieren
- Einmal geschrieben, mehrfache Nutzung
- Jeder kann Test laufen lassen
- Start auf Knopfdruck
- Schnell ablaufen

Definition eines guten Unit Tests

Ein Unit Test ist ein automatisiertes Stück Code, das eine zu testende Methode oder Klasse aufruft und dann einige Annahmen über das logische Verhalten dieser Methode oder Klasse prüft. Ein Unit Test wird fast immer mithilfe eines Unit Testing Frameworks erstellt. Er kann einfach geschrieben und schnell ausgeführt werden. Er ist vollständig automatisiert, vertrauenswürdig3, lesbar und wartbar.

Integration Tests



Bedeuted, dass zwei oder mehr voneinander abhängige Softwaremodule als gesamte Gruppe getestet werden.
Die Architektur von Applikation erfolgt über mehrere Komponenten und Ebenen, deshalb müssen Tests dies berücksichtigen.

Unit Test Beispiel

Wir haben eine Klasse SimpleParser mit einer Methode ParseAndSum, die Parameter erwaret. Diese Methode möchten wir nun testen.

```
public class SimpleParser
{
    public int ParseAndSum(string numbers)
    {
        if(numbers.Length==0)
        {
            return 0;
        }
        if(!numbers.Contains(","))
        {
            return int.Parse(numbers);
        }
        else
        {
            throw new InvalidOperationException(
            "Ich kann bisher nur mit den Zahlen 0 und 1 umgehen!");
        }
    }
}
```

Dazu schreibt man eine Produktionsklasse, die die Methode auf einen erwarteten Wert testet.

```
class SimpleParserTests
{
   public static void TestReturnsZeroWhenEmptyString()
   {
      try
      {
         SimpleParser p = new SimpleParser();
         int result = p.ParseAndSum(string.Empty);
         if(result!=0)
         {
            Console.WriteLine(
@"***SimpleParserTests.TestReturnsZeroWhenEmptyString:
-------
Parse and sum sollten für einen leeren String 0 zurückgeben");
         }
      }
      catch (Exception e)
      {
         Console.WriteLine(e);
      }
   }
}
```

Über eine Main Methode wird dann der Test gestartet.

```
public static void Main(string[] args)
{
    try
    {
        SimpleParserTests.TestReturnsZeroWhenEmptyString();
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
}
```

Junit - Mocks, Fakes, Stubs and Dummies

07 March 2015 16:17

Dummy

objects are passed around but never actually used. Usually they are just used to fill parameter lists.

Fake

objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).

Stubs

provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.

Mocks

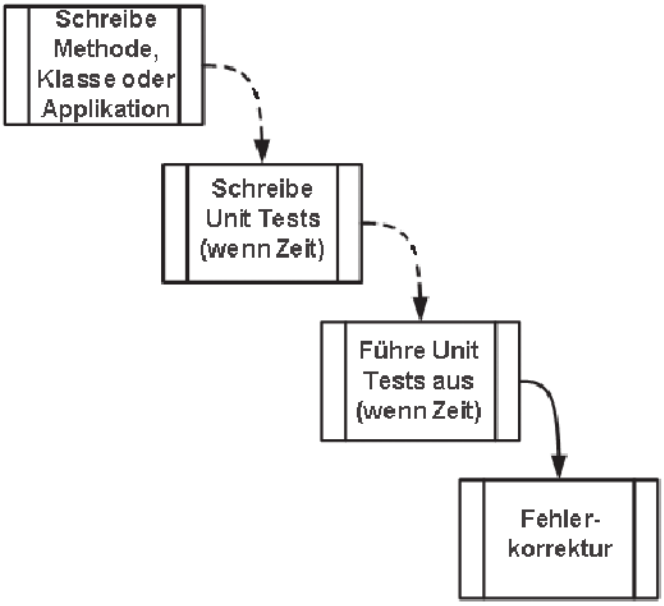
are what we are talking about here: objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

Testgetriebene Entwicklung

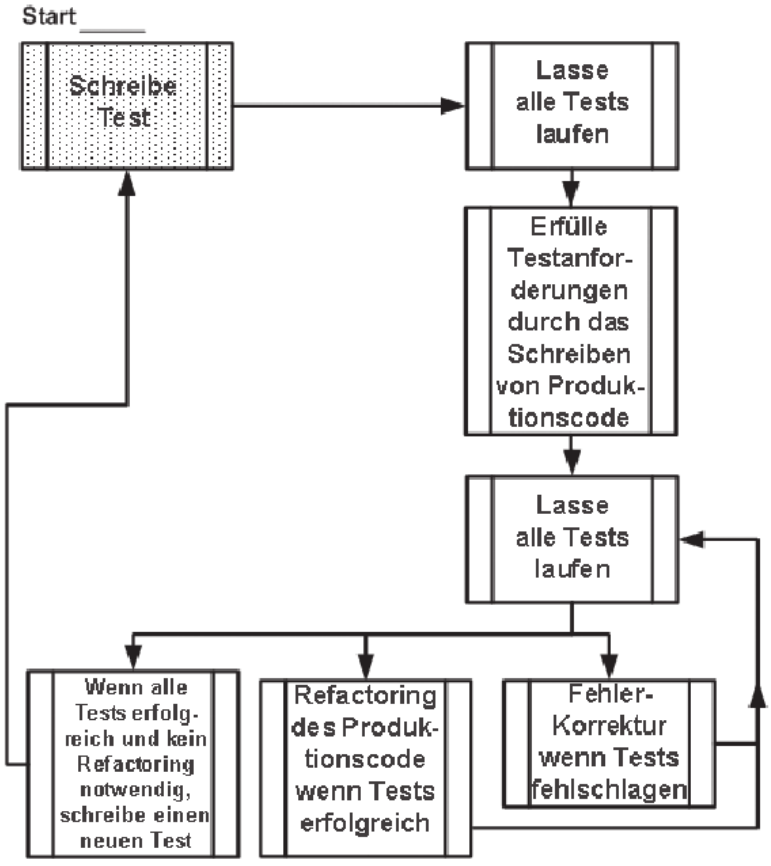
23 February 2015 13:50

Unterscheidet sich stark von der traditionellen Entwicklung.
Hierbei beginnt man mit einem Test, der fehlschlägt, schreibt und erweitert den Produktionscode bis der Test erfolgreich ist.

Traditionelle Weg:



Testgetriebene Entwicklung:



Der Prozess zur Korrektur des Codes nennt man Refactoring.
Refactoring ist im Grunde die Änderung eines code-Stücks, ohne die Funktionalität zu ändern.

UML

23 February 2015 14:06

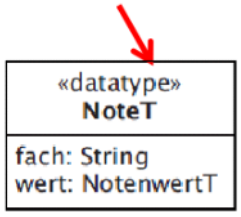
- Unified Modelling Language
- Ist eine Sprache für die Modellierung der Strukturen und des verhaltens von Software- und anderen Systemen

UML-Komponenten

Objekt

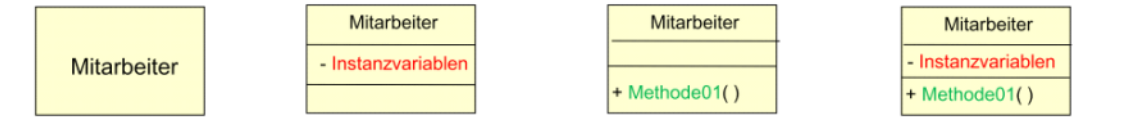
- Objekt ist immer Unterstrichen
- Ist nur die Klasse angegeben ist das Objekt anonym

Stereotyp



- <<...>>
- Ist eine Erweiterung der Modellelemente
- Gibt den Stereotyp des Objekts an
- Beispiel: <<enumeration>>

Klasse



nur mit Klassennamen

mit Klassennamen und Instanzvariablen

mit Klassennamen und Methoden

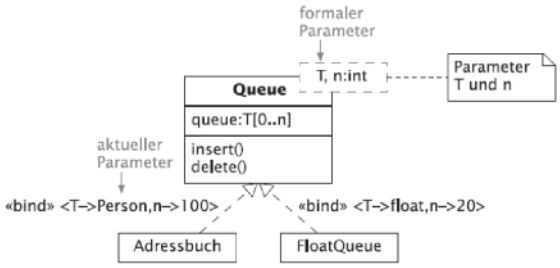
mit Klassennamen und Instanzvariablen Methoden

- Klassennamen muss innerhalb eines Pakets eindeutig sein
- Klassennamen werden nicht unterstrichten

Classifier

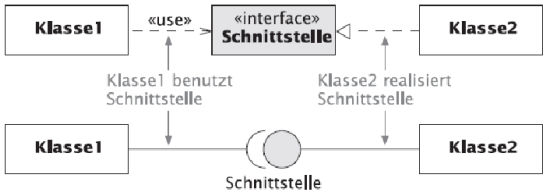
- Der Classifiert sprich klassierbar ist das oberste abstrakte und gedankliche Konstrukt in UML.
- Es ist eine Verallgemeinerung des Klassenkonzepts.
- Mit der UML Sprache kann UML selber beschrieben werden.

Parametisierte Klassen (Generics)



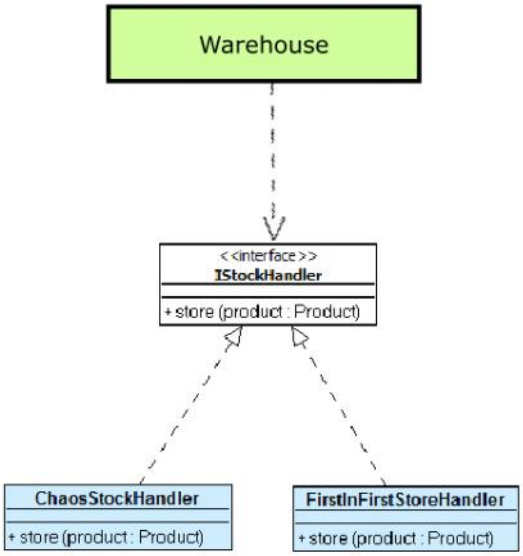
- Mit generics kann der erlaubte Datentyp angegeben werden.

Interface

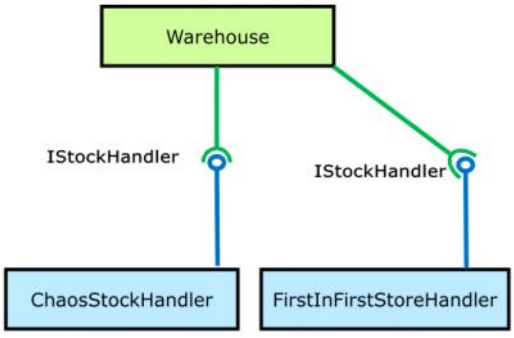


- Sind voll abstract, keine Realisierungen in einer Schnittstelle
- Beschreibt eine oder meherer Signaturen

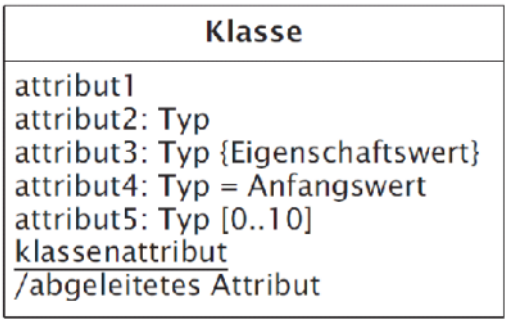
Beispiel: UML Ausführlich



Beispiel: UML Kurzschreibweise

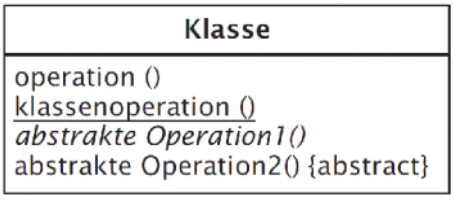


Attribute, Klassenattribute und Sichtbarkeit



- Beschreibt die Daten, die won Objekten oder Klassen angenommen werden können.
- Attribute werden spezifiziert mit: Typ, Mulitplizität, Anfangswert und Eigenschaftswert (readOnly, ordered, etc.)
- Klassenattribute werden unterstrichen.

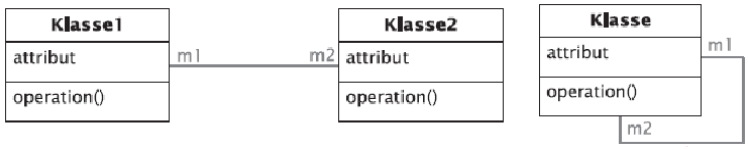
Operation und Methode



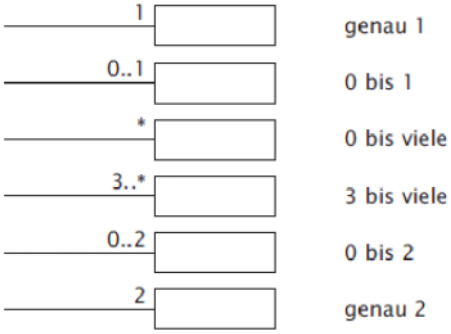
- Die Menge aller Operationen ist das Verhalten der Klasse

Assozationen

Die einfachste Form von Beziehungen, rechts eine **reflexive Assozation**.

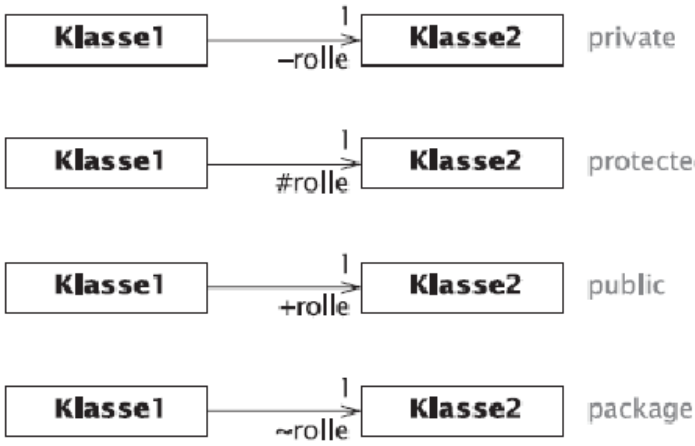


Multiplizität ist die Zahl, die definiert wie viele Objekte mit einem anderen Objekt verbunden werden.

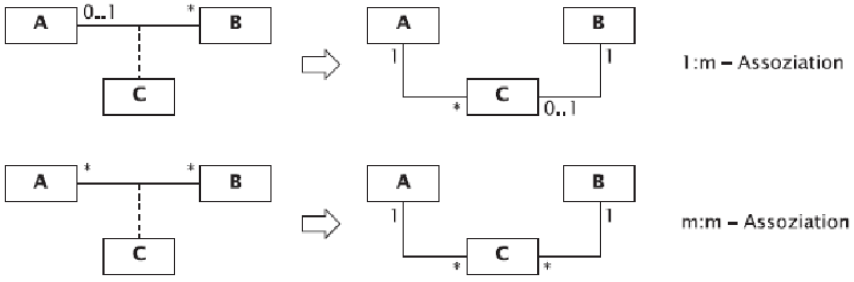


Jeder Professor liest **0 bis viele** Vorlesungen.
Jede Vorlesung **wird** von **genau 1** Professor gelesen.

Zusätzlich kann zu jeder Beziehung eine Rolle angegeben werden.

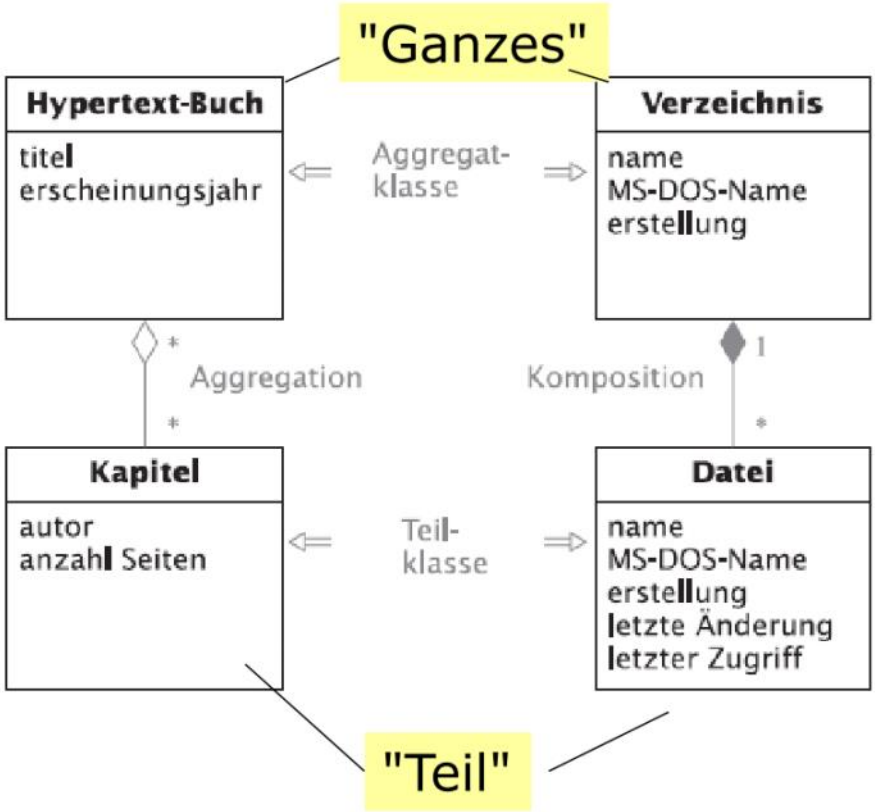


n:m - Beziehungen müssen speziell dargestellt werden. Die Assoziation wird über eine dritte/zustätzliche Tabelle dargestellt.



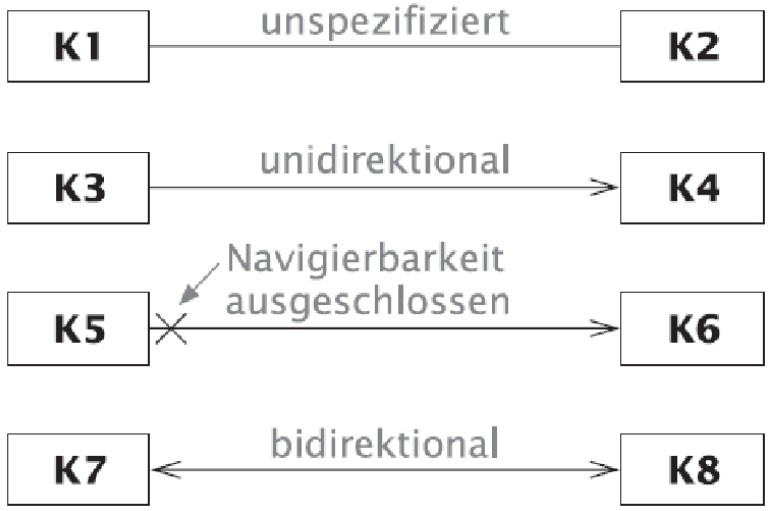
UML unterscheidet zwischen drei Arten von Assoziation.

- Einfache Aggregation -> eine einfache Beziehung
- Aggregation -> Teil von
- Komposition -> Abhängig von



Einzelne Beziehungen können um die Navigierbarkeit ergänzt werden, d.h. es wird mittels Pfeil

angegeben in welche Richtung das Objekt seine Fremdobjekte aufrufen kann.



Generalisierung / Abstrakte Klasse

Generalisierung ist eine gerichtete Beziehung zwischen einem speziellen und einem generellen Classifier.

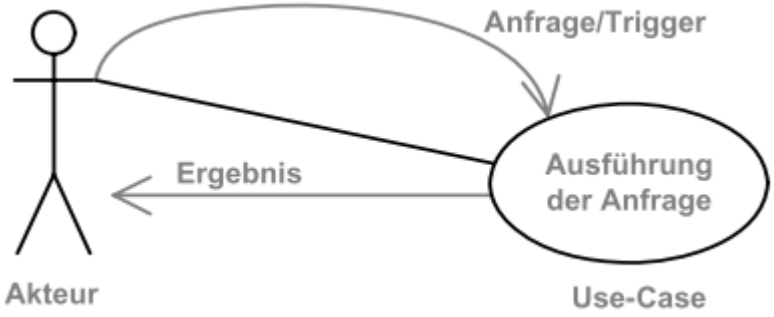
- Basis(Super-, Ober-, Eltern-)Klasse: Klasse1
- Abgeleitete (Sub-, Unter-, Kind-)Klasse: Klasse 2 und 3
- Ableitung, Spezialisierung: Vorgang des Erbens

UML Use-Case

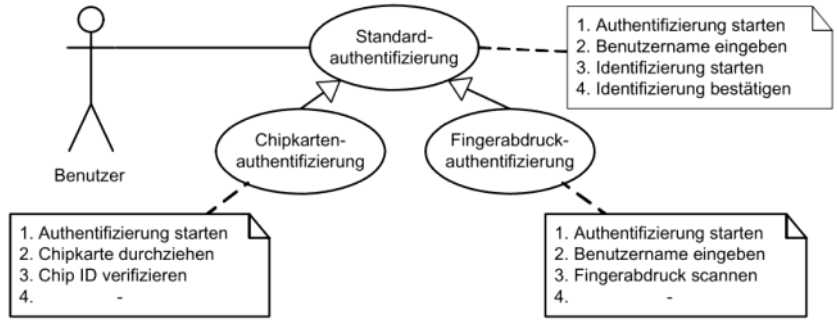
26 February 2015 14:42

Eine Use-Case beschreibt eine Menge von Aktionen, die schrittweise ausgeführt, ein spezielles Verhalten formen.

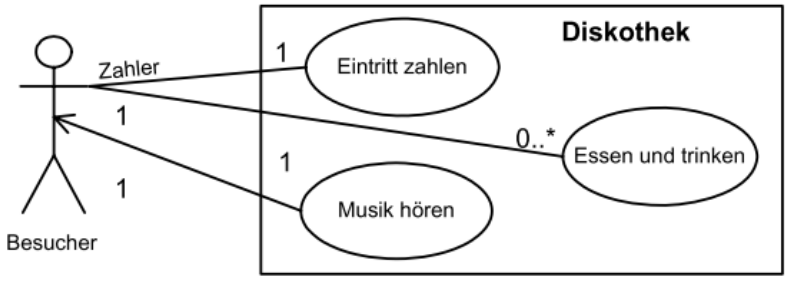
Der Use-Case wird von Aktuer ausgeführt. Ganz allgemein sieht das so aus:



Dazu eine Beispiel Implementierung.



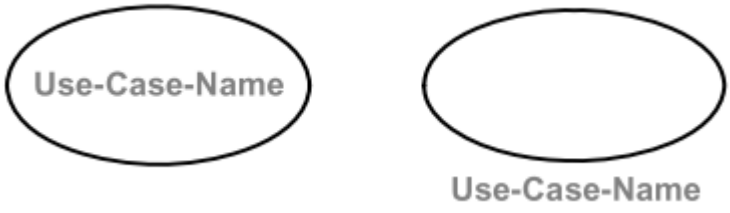
Das Diagram kann auch mit bekannten Elmenten der Assozation und Multiplizität erweitert werden.



Elemente

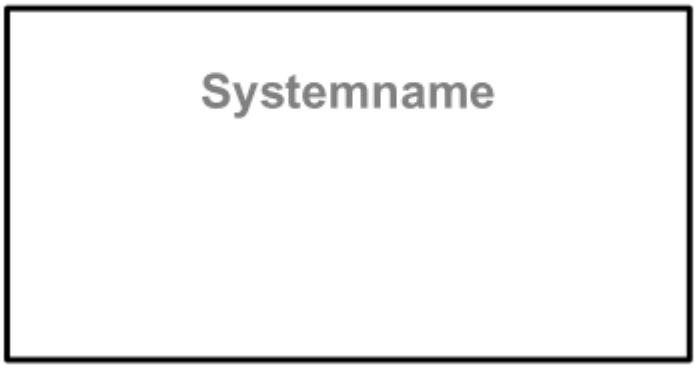
Use-Case

Beschreibt eine Aktion.



System

Stellt die Systemgrenze des Use-Case dar.

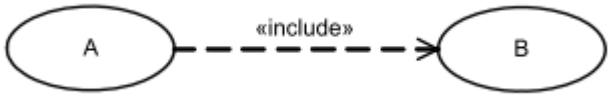


Akteur

Der Akteur interagiert mit dem System, natürlich immer von ausserhalb.

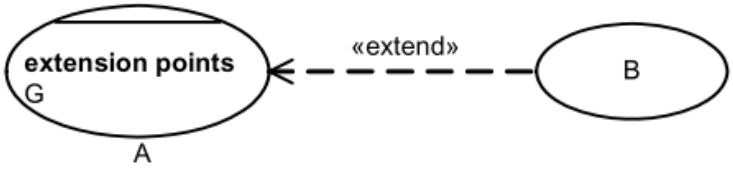
include

Die include-Beziehung visualisiert, dass ein Use-Case A das Verhalten eines anderen Use-Case B importiert.



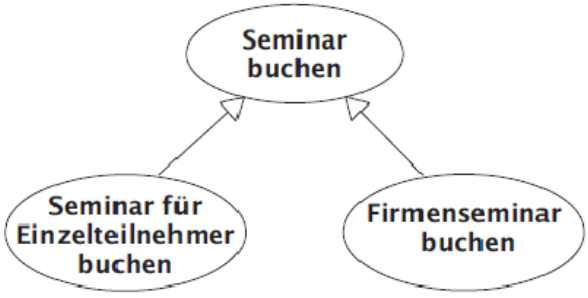
extend

Die extend-Beziehung zeigt and, dass das verhalten eines Use-Case A durch einen anderen Use-Case B erweitert werden kann, **aber nicht muss**.

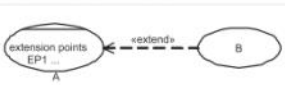


is a

Dient zur Generalisierung der Cases

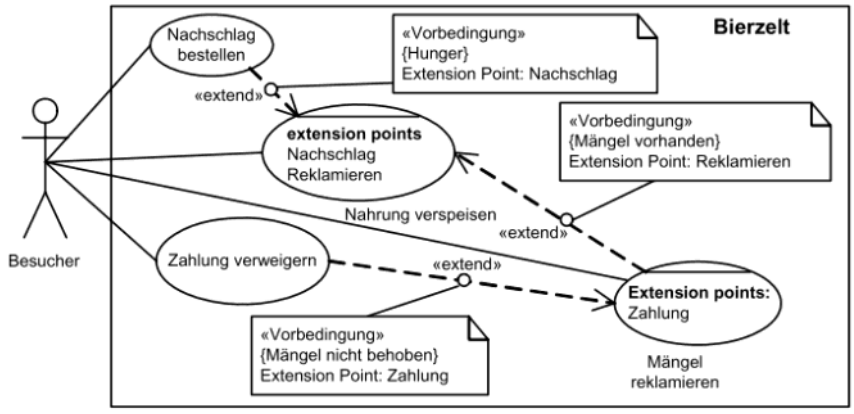


include und extend Vergleich

	«include»-Beziehung	«extend»-Beziehung
Notation		
Bedeutung	Ablauf von A schließt immer Ablauf von B ein.	Ablauf von A kann, muss aber nicht durch Ablauf von B erweitert werden.
Wann wird die Beziehung genutzt?	Ablauf von B kann in verschiedenen Use-Cases genutzt werden.	A besitzt neben Normalverhalten auslagerbare Sonderfälle.
Bedeutung für die Modellierung	A ist meist unvollständig und wird erst durch Inklusion B zu einem vollständigen Ablauf.	A ist meist vollständig und kann durch B optional erweitert werden.
Abhängigkeiten	A muss B bei der Modellierung berücksichtigen. B wird **unabhängig von A** modelliert, um die Nutzung durch weitere Use-Cases sicherzustellen (Wiederverwendbarkeit), B muss in sich **nicht vollständig** sein („B weiß nicht, durch wen er inkludiert wird“).	A muss durch Angabe von Erweiterungspunkten auf die Erweiterung durch B vorbereitet werden. B wird **in sich vollständig** und **unabhängig** von A modelliert („B weiß nicht, wen er erweitert“).

Beispiel extend

Mänge wie Nachschlag oder Reklamieren sind optional.



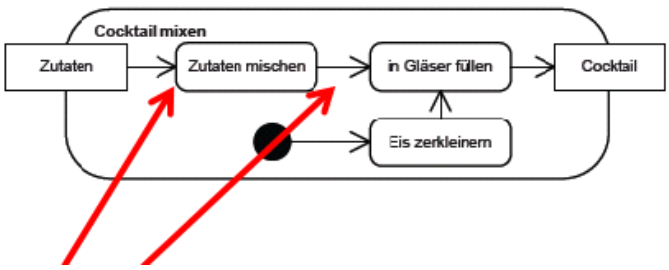
UML Aktivitätsdiagram

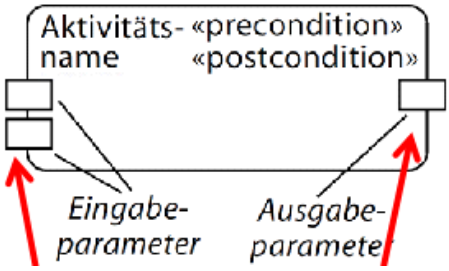
02 March 2015 14:30

Dienen zur Beschreibung eines Ablaufs von Aktivitäten.

Anwensungsmöglichkeiten:

- Geschäftsprozessmodellierung
- Beschreibung von Use-Cases
- Implementierung von Operationen





Elemente

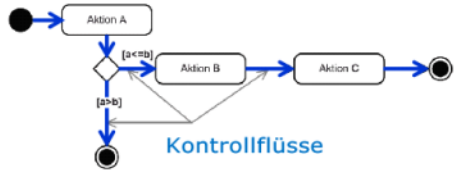
Kante (Acitivity Egde)

Übergang zwischen zwei Aktionen, Objektkanten, Kontrollelemente

Man unterscheided zwischen Kontrollfluss und Objektfluss

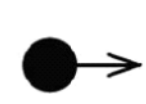
Kontrollfluss:
Beschreibt eine Kante zwischen zwei Aktionen (B zu C) oder zwischen Aktion und Kontroll-elementen (z.B. A zu Verzeigung) oder....?
Token auf diesen Kanten stimulieren "nur" die Ausführung einer (nächsten) Aktion.

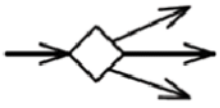
Objektfluss:
An einem Objektfluss ist mindestens ein Objektknoten beteiligt. Die Kante über-trägt Token, welche selber Daten/Objekte zum oder vom Objektknoten transportieren.

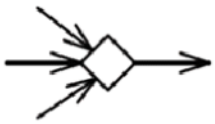


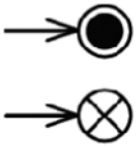
Kontrollelemente

Steuert den Ablauf der Aktivität.


Startknoten

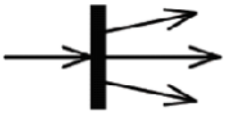

Verzweigungsknoten

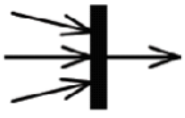

Verbindungsknoten


Endknoten

Ende der Aktivität (beendet gesamte Aktiviät)

Ende des Kontrollflusses (beendet einen einzelnen Ablauf)


Parallelisierungsknoten


Synchronisationsknoten

Objektknoten

Repräsentieren beteiligte daten oder Gegenstände der realen oder vorstellbaren Welt.

① In Objektknoten hineingehende Token (folgende Seiten) repräsentieren "Daten" d.h. Werte / Objekte

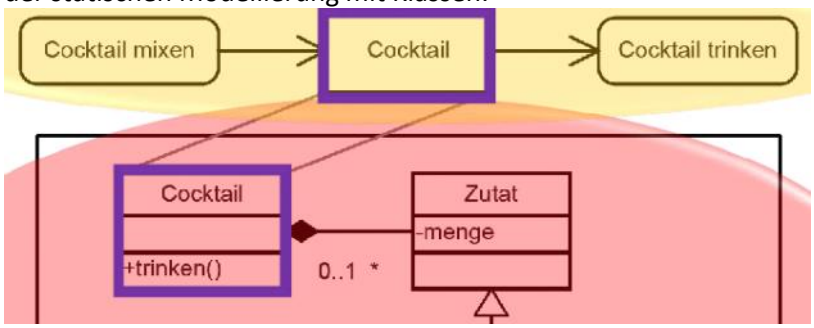
② Aus den Objektknoten herausgehende Token transportieren Daten (Werte / Objekte) an die Folgeaktion.

Token

Zeigt and wo sich der Ablauf einer Atkivität befindet.

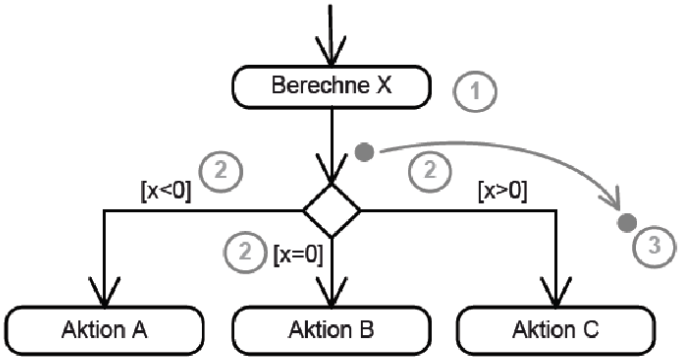
Objektknoten - Bindeglied

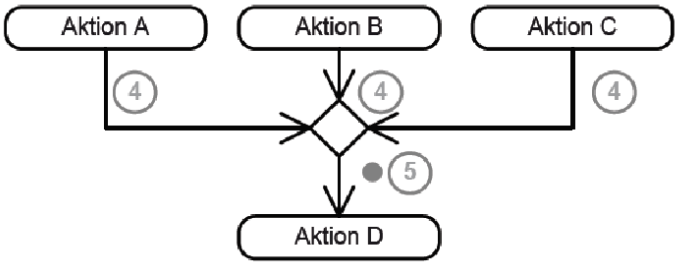
Objekteknoten sind auch Bindeglieder zwischen der dynamischen Modellierung mit Aktivitäten und der statischen Modellierung mit Klassen.



Token: Alternativen/Entscheide

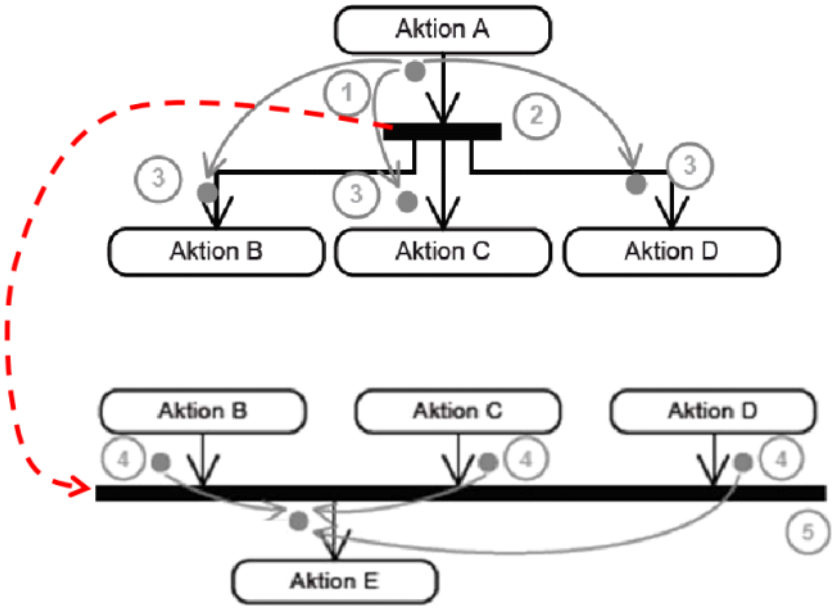
Der Ablauf erfolgt zwingend nur durch eine Aktion. Die Entscheidung wird am Kontrollknoten gefällt.





Token: Parallelabarbeitung

Der Ablauf wird geteilt und parallel fortgeführt. Die Auteilung wird am Kontrollknoten gemacht und zusammengeführt.



Signalsender

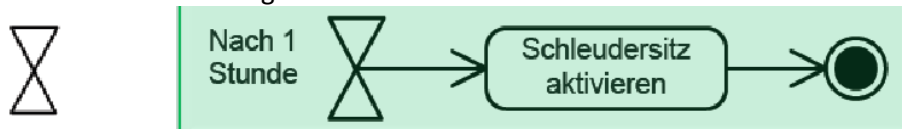
Sended Signal an Ereignisempfänger.

Ereignisempfänger

Gegenstück zum Signalsender.

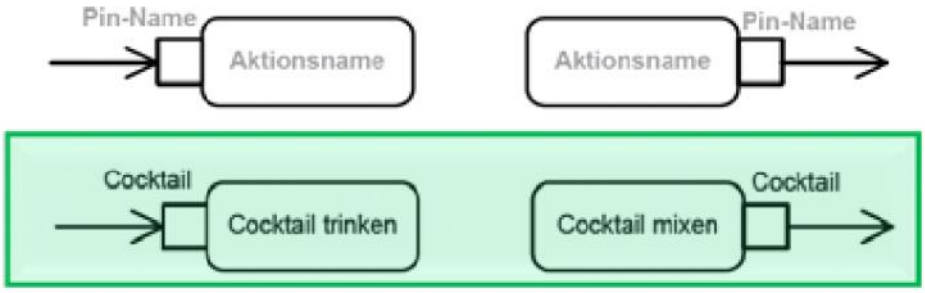
zeitabhängiger Auslöser

Löst Aktion nach einer gewissen Zeit aus.



Pin-Notation

Stellt Zusammenhang zwischen Aktion und Objektknoten als Eingabe- bzw. Ausgabeparameter der Aktion dar.



Sprungmarken

Sprung zu Aktionen.

UML Interaktionsiagramme

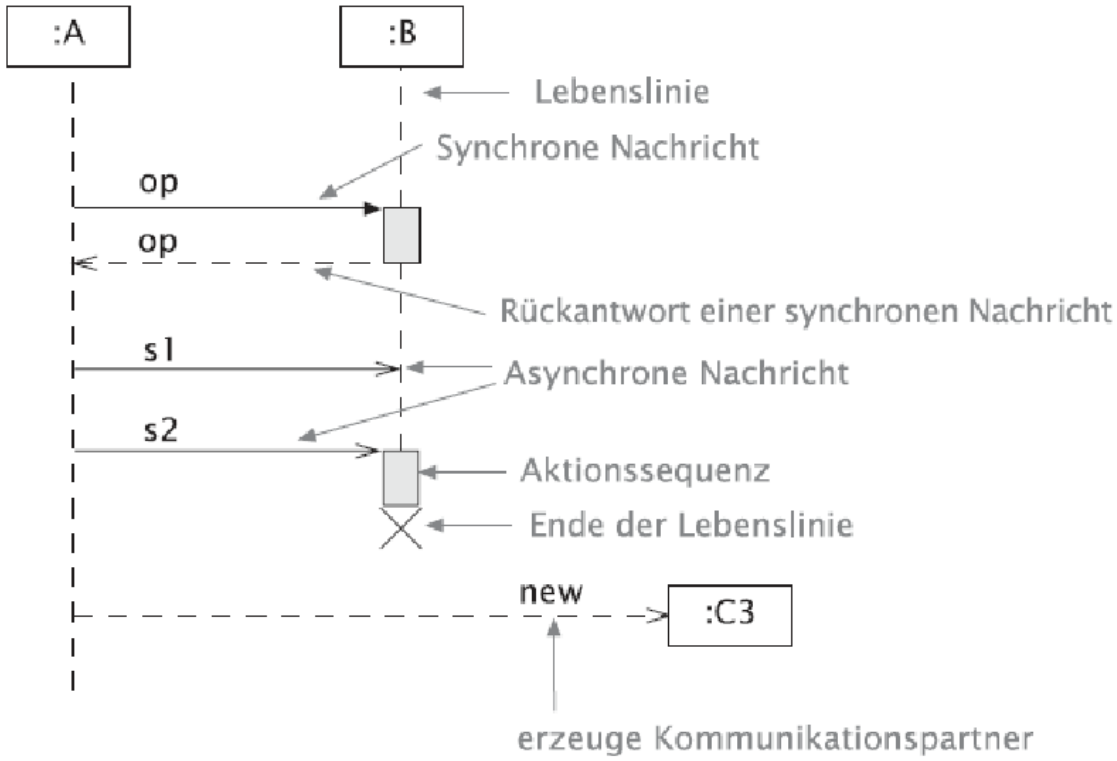
02 March 2015 16:00

Sequenzdiagramm

Zeigt die Interaktion zwischen mehreren Kommunikationspartner (Objekte)

Anwendungsmöglichkeiten:

- Interaktionen auf verschiedenen Abstraktionsebenen
- Struktur einer Klasse
- Komponente
- System
- Protokollkommunikation

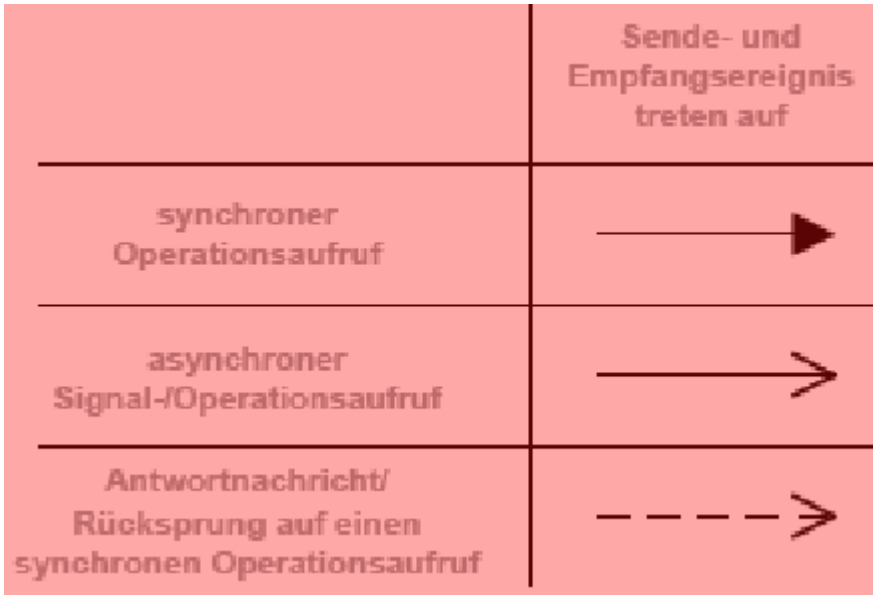


Der Austausch von Nachrichten wird als Interaktion beschrieben. Der zeitliche Ablauf wird durch die Lebenslinie definiert.

Elemente

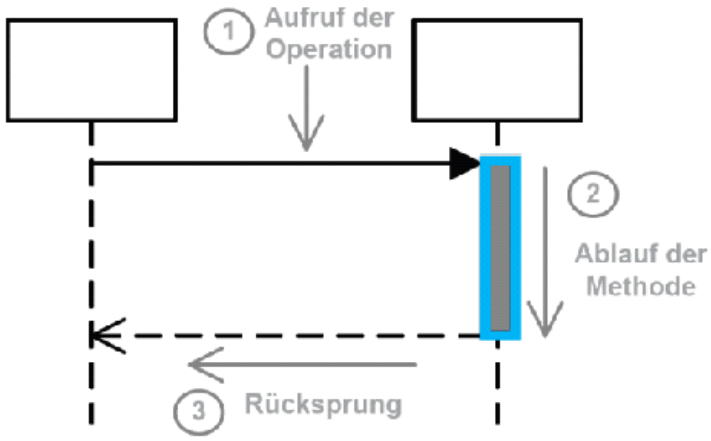
Nachrichten

Repräsentiert den Informaitonsfluss zwischen Kommunikationspartnern.



Ausführungssequenz

Ist die Tätigkeit, die beim Empfangen von Nachrichten ausgelöst wird.

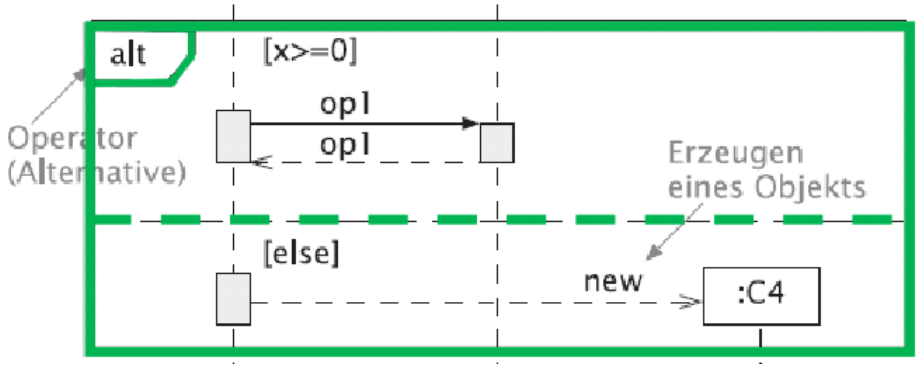


Fragmente

Beschreibt den Teil einer Interkation für den bestimmte Regeln gelten.
Dazu gibt es zwölf Interaktionsoperatoren.

Deutsche Bezeichnung	Englische Bezeichnung	Kürzel im Diagramm	Damit modellieren Sie ...
Alternative Fragmente	Alternative	`alt`	... alternative Ablaufmöglichkeiten
Optionales Fragment	Option	`opt`	... optionale Interaktionsteile
Abbruchfragment	Break	`break`	... Interaktionen in Ausnahmefällen
Negation	Negative	`neg`	... ungültige Interaktionen
Schleife	Loop	`loop`	... iterative Interaktionen
Parallele Fragmente	Parallel	`par`	... nebenläufige Interaktionen
Lose Ordnung	Weak Sequencing	`seq`	... von Lebenslinie und Operanden *abhängige* chronologische Abläufe

Un im diagramm sieht das so aus:

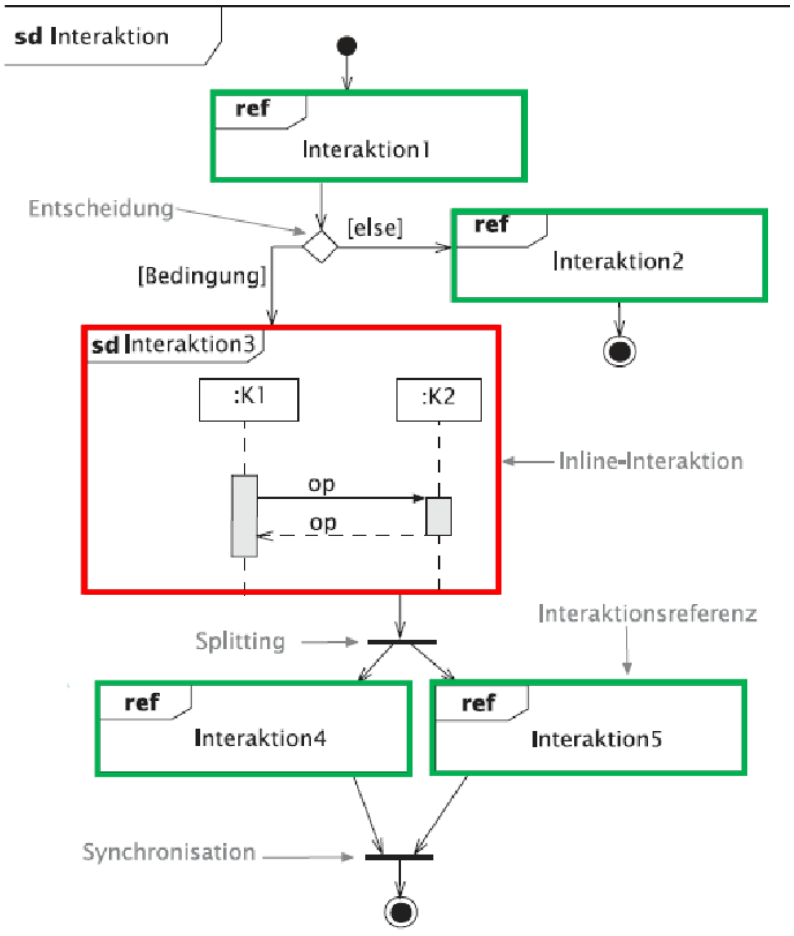


Kommunikationsdiagramme

Ist genau dasselbe wie ein Sequenzdiagramm, wird einfach anderst dargestellt.

Interaktionsübersichtsdiagramme

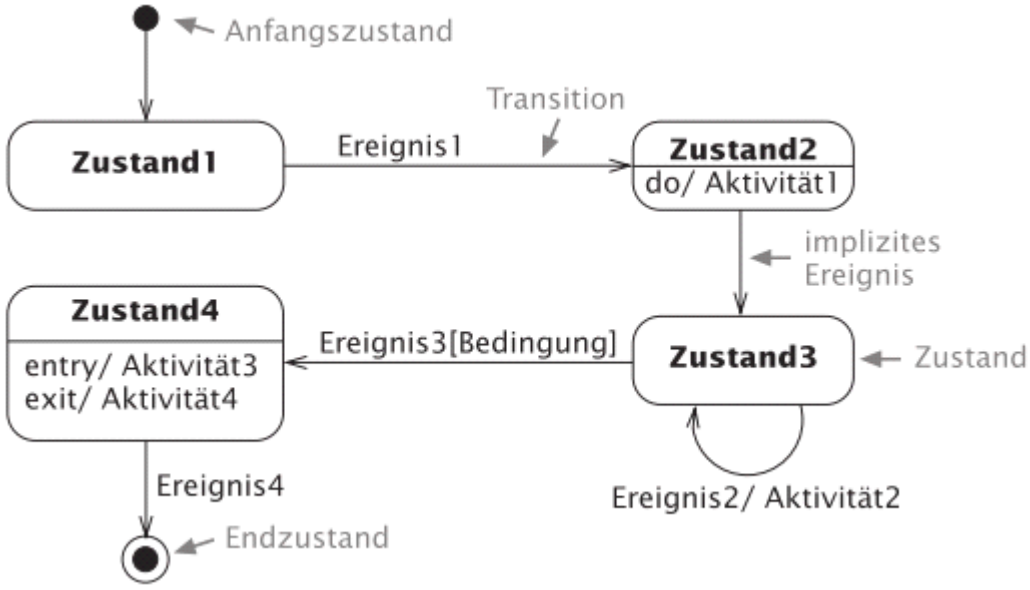
Ist eine Variante des Aktivitätsdiagramms, bei der jeder Knoten eie Inline-Interaktion (sd I) oder eine Interaktionsreferenzbeschreibt (ref).



UML Zustanddiagramme

02 March 2015 16:19

Ein Zustand ist eine Zeitspanne, in der ein Objekt auf ein Ereignis wartet.
Das Objekt verweilt eine bestimmte Zeit in einem Zustand bis es durch ein entsprechendes Ereignis geht.
Ereignisse sind Zeitpunkt getrieben und haben keine Dauer.



Elemente

Transition

Verbindet zwei Zustände.
Wird durch ein Eregnis ausgelöst.
Eine Transition vollführt einen Zustandswechsel.

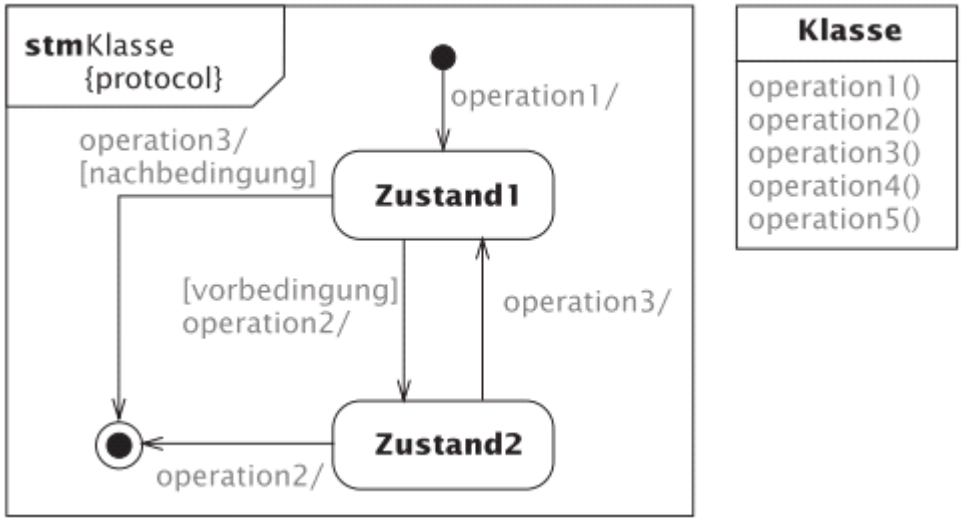
Ereignis

Kann sein:

- Signal: Übermitteln asnychrone Nachrichten
- Operationsaufruf: Ruft Operation beim Empfänger auf
- zeitliches Ereignis: Verstrichene Zeitspanne oder bestimmter Zeitpunkt
- Änderung bestimmter Werte: Veränderung bestimmter Wert löst Ereignis aus
- implizietes Ereignis

Können and Bedingungen geknüpft werden.

Beispiel - Implementation einer Klasse:


stmKlasse
{protocol}
operation1/
operation3/
[nachbedingung]
Zustand1
[vorbedingung]
operation2/
operation3/
Zustand2
operation2/

Klasse
operation1()
operation2()
operation3()
operation4()
operation5()

Softwarearchitektur

05 March 2015 13:48

- 1969: Grüdnung ICSE - Internationale Software-Engineering Konferenz.
- Requirements Engineering und Businessanalyse im Vordergrund
- Lerneinheit zeigt mit pragmatischen Ansätzen schnellere und bessere kontstruktion von Systemarchitekturen.

Begriff

- Architektur beschreibt die Struktur des Systems
  - Bausteine
  - Schnittstellen
  - Beziehungen
- Software-Architektur ist Plan des Systems
- Gute Software-Architektur
  - relevant
  - effizient pflegbar
  - sparsam
  - verständich und nachvollziehbar
  - korrekt und aktuell
- Als Software-Architekt konstruiert man
  - langfristig -> kontinuierliche weiterentwicklung
  - wartbar -> WenigerAufwand
  - flexibel -> Erweiterbarkeit
  - verständlich -> saubere Dokumentation

Template Dokumentation

Unter http://www.arc42.com findet man eine Übersicht zur Dokumentation.

1. Einführung und Ziele
Aufgabenstellung, Qualitätsziele, eine Kurzfassung der architekturrelevanten Anforderungen (insb. die nichtfunktionalen), Stakeholder.

2. Randbedingungen
Welche Leitplanken schränken die Entwurfsentscheidungen ein?

3. Kontextabgrenzung
In welchem fachlichen und/oder technischen Umfeld arbeitet das System?

4. Lösungsstrategie
Wie funktioniert die Lösung? Was sind die fundamentalen Lösungsansätze?

5. Bausteinsicht
Die statische Struktur des Systems, der Aufbau aus Implementierungsteilen.

6. Laufzeitsicht
Zusammenwirken der Bausteine zur Laufzeit, gezeigt an exemplarischen Abläufen ("Szenarien")

7. Verteilungssicht
Deployment: Auf welcher Hardware werden die Bausteine betrieben?

8. Querschnittliche Konzepte und Muster
Wiederkehrende Muster und Strukturen.
Fachliche Strukturen.
Querschnittliche, übergreifende Konzepte, Nutzungs- oder Einsatzanleitungen für Technologien. Oftmals projekt-/systemübergreifend verwendbar!

9. Entwurfsentscheidungen
Zentrale, prägende und wichtige Entscheidungen.

10. Qualitätsszenarien
Qualitätsbaum sowie dessen Konkretisierung durch Szenarien

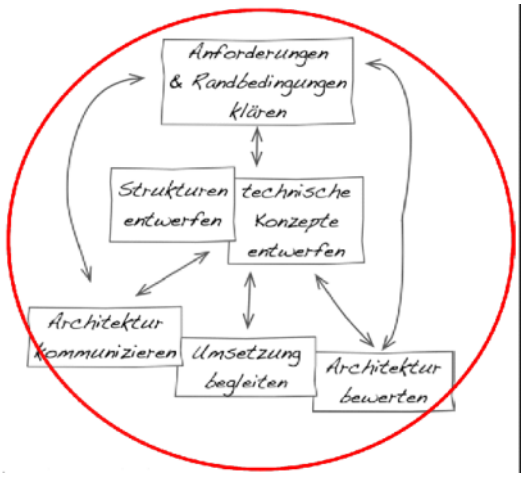
11. Risiken

12. Glossar
Wichtige Begriffe.

Legende:

anforderungsbezogene Informationen	Strukturen der Lösung (Sichten)	übergreifende (technische) Informationen	besonders wichtige Entscheidungen

Der Architekt (TvA)



- Dokumentiert in jedem Schritt die expliziten Architektur-Ziele.

Architekturziele VS **Projektziele**:

- Architekturziele stellen den

- Projektziele ermöglichen zu

Aufgaben

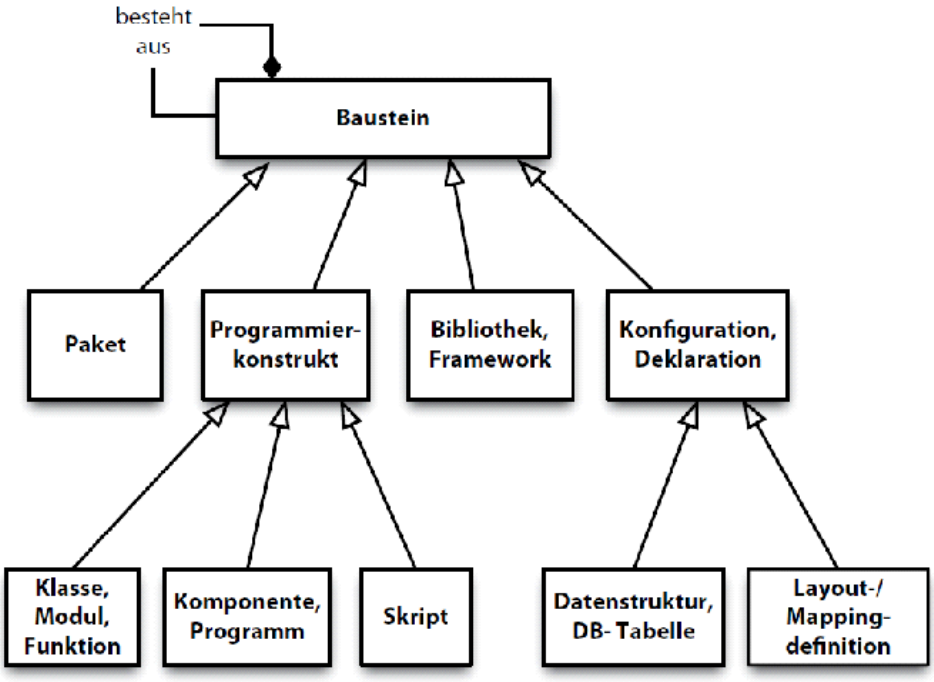
- Entwerfen Strukturen des Systems
- Technische Entscheide
  - z.B. für lokalen Baustein
  - Allgmeine Entscheide zentral und redundanzfrei dokumentieren
  - Top-Level-Strukturen
  - Qualitätsmerkmale beeinflussen
  - Kostenrelevanz definieren
  - Bewusstsein externe Schnittstellen
  - Auswirkung Systembetrieb bekannt
  - Risikobewusstsein
  - Technologieentscheide treffen
- Vereinfachen können
  - Einfachheit von Entwürfen und Systemen
  - Überblick behalten
- Implementieren können
- Nachvollziehbar für Projekt dokumentieren
  - systematisch und methodisch
  - Balance zwischen zu viel und zu wenig finden
- Kommunizieren können
- Schätzen und bewerten können
  - Aufwand und Kosten
- Balancieren können
  - diplomatische Fähigkeiten
  - Trade-offs erkennen > Qualitätsziele
- Beraten können
- Vermarkten können

SA Bausteine

05 March 2015 15:01

Meta-Modell

Definiton von verschiedenen Bausteinen/ Kompoenenten.



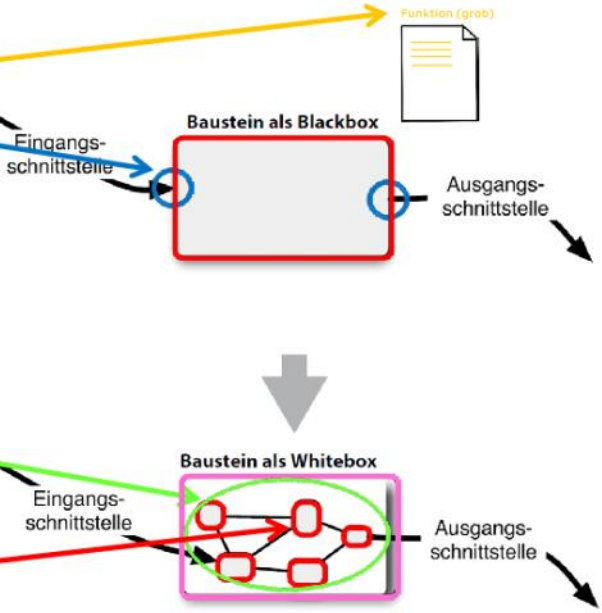
Es gibt

- Programmierkonstrukte
  - Klassen
  - Module
  - Funktionen
  - Skripte
  - Programme
  - Komponenten
- Pakete
  - Namensräume
  - Gruppierung von Konstrukten
- Konfiguration
  - Bestimmt Verhalten oder die Schnittstelle von Softwareteilen
  - Stylesheets
- Bibliotheken, Frameworks und Standardsoftware

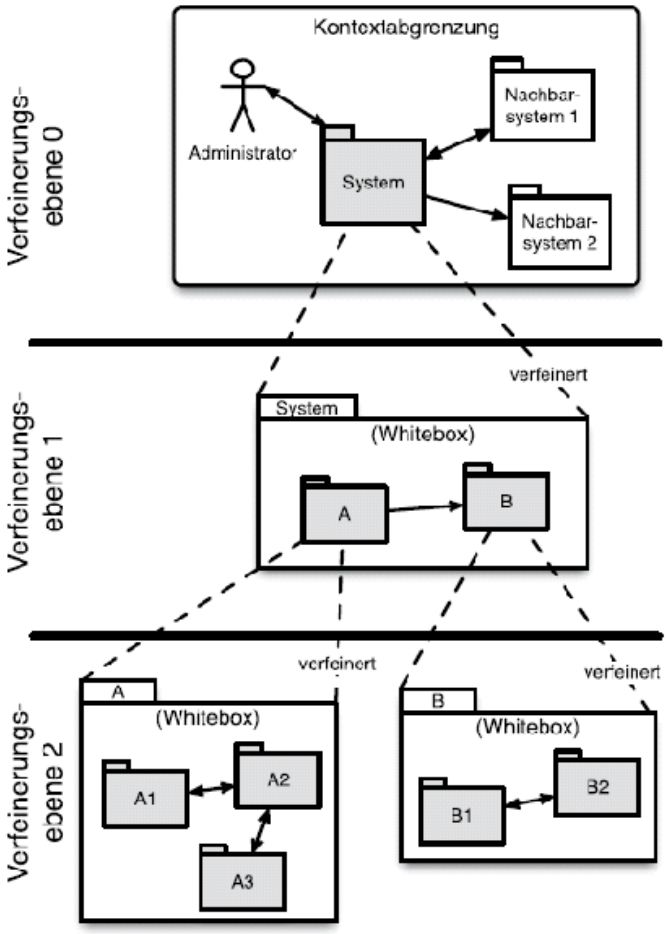
Blackboxes und Whiteboxes

Blackboxes sind nur durch ihre Schnittstellen und Funktionen charakterisiert. Sie verstecken ihr gesamtes Innenleben, ihre inneren Strukturen und Abläufe. Somit eignen sich Blackboxes zur Vereinfachung, zur Abstraktion und als Überblick. Sie dokumentieren Schnittstellen-absprachen, die bei weiterer Präzisierung eingehalten werden müssen.

Whiteboxes zeigen die innere Struktur von Bausteinen. Sie stellen „geöffnete" Blackboxes dar und bestehen ihrerseits wiederum aus einer Menge von (kleineren) Blackboxes mit deren Beziehungen und Abhängigkeiten.



Verfeinerung von Definitionen



Verfahren:

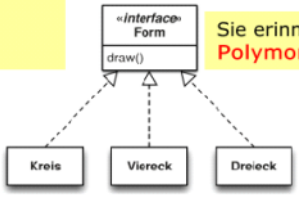
- Zerlegen der Software in Bausteine und Beziehungen
- Die Abhängigkeit von Bausteinen voneinander > Koppelung
  - Aufruf von Funktion/Methode eines Bausteins
  - Erzeugung von Bausteinen
  - Daten oder Datenstrukturen über Parameterangaben
  - Hardware oder Laufzeitumgebung z.B. Adressraum, virtuellen Maschinen oder Netzwerksegmenten > Web-Services

- Abhängigkeit verringern
  - Koppelung mus lose sein
  - Kohäsion eine Bausteins hoch halten
  - Komponenten off für Erweiterungen definieren
  - ...schlecht:

```
...
void draw(Form f) {
    if (f.type == Kreis)
         drawCircle( f );
    else if (f.type == Viereck)
         drawSquare( f );
...
}
```

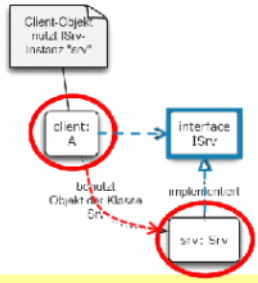
Ziel: Wir wollen neue Form "Dreieck" aufnehmen...
=> wie? .....??

...besser:



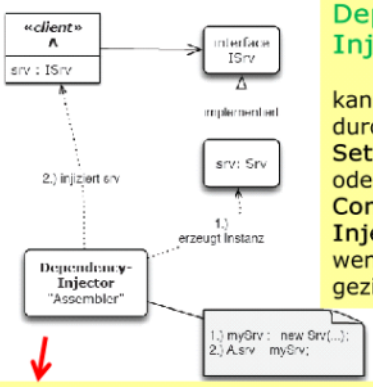
Sie erinnern sich? Polymorphismus!

  - Bevorzugen Abhängigkeiten von Abstraktion statt von Implementierungen
  - Mehrere spezifische Schnittstellen statt eine grosse Universalschnittstelle
  - Zyklische Abhängigkeiten vermeiden -> Paradoxe Probleme
  - Klassen müssen bei der Vererbung durch Unterklassen ersetzbar sein
  - Bei Nutzung von Schnittstellen entsteht das Problem zur Laufzeit einer Schnittstelle eine Instanz zu beschaffen -> Wer übernimmt den Lifecycle dieser Instanz?
  - Beispiel: ...schlecht:



Wenn A selber verantwortlich ist für die konkrete Instanz von srv:Srv ist das eine enge Koppelung ... das wäre nicht gut!

...besser:



z.B. Spring, PicoContainer, Guice,...

Über das Entkoppelungsmuster **Dependency Injection**

kann man z.B. durch **Setter-Injection** oder **Constructor-Injection** die notwendigen Objekte gezielt aufrufen

Remote Procedure Calls (RPC)

16 March 2015 12:09

Das Konzept von RPC wurde ursprünglich vom US-Unternehmen Sun Microsystems für das so genannte *Network File System (NFS)* entwickelt. Dieses Dateisystem sollte, wie der Name schon besagt, Speicher über ein Netzwerk zur Verfügung stellen können, und dabei so zuverlässig wie ein stationäres Dateisystem sein. Zu diesem Zweck wurde mit RPC eine Sammlung von Befehlen entwickelt, die nach dem Schema eines Client-Server-Modells (siehe hierzu auch Host-Architekturen) diese Speichervorgänge erledigen konnten.
Grundsätzlich funktioniert die RPC-Kommunikation anhand einer einfachen Architektur:

1. Die Client-Anwendung sendet eine RPC-Nachricht zunächst an den Server. Hierzu ist auf dem Server ein Dienst (der so genannte *Portmapper* oder *Endpointmapper*) installiert, der die RPC-Anfragen zentral empfängt, die in der RPC-Nachricht hinterlegten Informationen über die gewünschte Anwendung ausliest und an die entsprechende Server-Anwendung leitet.

2. Die Server-Anwendung erhält die RPC-Nachricht vom RPC-Dienst, verarbeitet entsprechend die Information und sendet seine Antwort an den Client zurück.

Äußerst wichtig ist bei einer solchen Kommunikation über ein Netzwerk die so genannte*Fehlersemantik*, also die Reaktion auf Kommunikationsfehler. Dies muss der Programmierer einer Anwendung unbedingt berücksichtigen, um beispielsweise bei einer fehlenden Server-Antwort auf eine RPC-Nachricht diesen "hängenden" Zustand abzufangen. Gleichzeitig muss unbedingt vermieden werden, dass wiederholt gesendete RPC-Nachrichten beim Server auch als solche erkennbar sind, damit dieser Vorgänge nicht doppelt ausführt. Es gibt deshalb in RPC verschiedene Möglichkeiten, solche Zustände und wiederholende Nachrichten zu definieren, die zur Gewährleistung der Datenintegrität von Programmierern auch zwingend genutzt werden.
Unterschieden wird bei der RPC-Verarbeitung zwischen *synchronem RPC* und *asynchronem RPC*. Bei synchronem RPC muss die Verarbeitung genau Schritt für Schritt erfolgen. Das bedeutet, dass der Client eine RPC-Nachricht an den Server sendet und zwingend auf die Antwort warten muss, bevor er weiter fortsetzen kann. Bei Anwendungen, die mit asynchronem RPC kommunizieren, kann der Client nach dem Senden einer RPC-Nachricht weitere Operationen vornehmen, da auch eine späterer Empfang der Server-Antwort einkalkuliert ist. Diese Funktionsweise hat durchaus seine zentrale Bedeutung, denn Anwendungen, die mit synchronem RPC kommunizieren, ist es sehr wichtig, dass die Anwendung eine höhere Laufzeit der Server-Antwort berücksichtigt, da sonst unter Umständen bei höheren Wartezeiten die Anwendung schlicht stehen bleibt und für den Benutzer unangenehme Wartezeiten und der Eindruck eines "hängenden Programms" entsteht (was an sich dann auch tatsächlich der Fall ist). Andererseits erfordern Anwendungen mit asynchronem RPC in der Regel einen höheren Programmieraufwand.
Ärgerlicherweise gibt es mehrere RPC-Implementierungen, so dass nicht von "einem" RPC-Standard gesprochen werden kann:

- **Implementierungen nach Sun RPC**
  Die ursprüngliche Implementierung wurde von Sun Microsystems entwickelt und liegt inzwischen in zwei eigenständigen Modellen vor, die beide von Sun RPC abgeleitet sind. Beide Modelle nutzen standardmäßig den TCP- und UDP-Port 111.

  - **Transport Independent Remote Procedure Call** (TI RPC)
    TI RPC wird hauptsächlich mit dem Unix-Betriebssystem Solaris genutzt und ist eine deutliche Weiterentwicklung, die nicht nur mit TCP und UDP funktioniert, sondern auch mit anderen Transportprotokollen.

  - **Open Network Computing Remote Procedure Call** (ONC RPC)
    Die ONC-RPC-Implementierung ist eine, wie der Name schon sagt, offene Implementierung, die sich beispielsweise im offenen Betriebssystem Linux findet.

- **Distributed Computing Environment Remote Procedure Call** (DCE RPC)
  Diese RPC-Implementierung basiert auf eine frühe RPC-Version, die später von der *Open Software Foundation (OSF)* als Basis für diese Implementierung genutzt wird. Auf DCE RPC basiert auch die RPC-Implementierung von Microsoft, die sich *MSRPC* nennt. Im Gegensatz zu den obigen RPC-Implementierungen nutzen DCE-RPC-Implementierungen den TCP- bzw.

UDP-Port 135 zur IP-Kommunikation.

- **ISO Remote Procedure Call** (ISO RPC)
  ISO RPC stellt einen Versuch dar, einen einheitlichen RPC-Standard zu etablieren. Mangels Implementierungen und auch wegen einigen fehlenden Features ist ISO RPC jedoch kaum verbreitet und spielt derzeit keine signifikante Rolle.

RPC via XML

Als universelles Datenaustauschformat ist auch XML, die *Extensible Markup Language* (siehe hierzu auch XML - Extensible Markup Language), prädestiniert für die Übertragung von Funktionsaufrufen. Eine solche einheitliche Übertragung ist zum Beispiel in Web-Anwendungen sinnvoll, die über so genannte Webservices mit anderen Systemen kommunizieren können. So können beispielsweise fremde Anwendungen über Programmschnittstellen, so genannten *API*(*Application Programming Interface*) verhältnismäßig einfach in andere Web-Anwendungen integriert werden. Die Abbildung dieser Anfragen in XML-Abfragen erleichtert die Programmierung und Kommunikation solcher Anwendungen gewaltig. Einleuchtend dabei ist auch, dass RPC und XML eine nahezu perfekte Ehe eingehen könnten, um Remote-Befehle und deren Ergebnisse via XML zu übertragen.
Ein erster Schritt in diese Richtung wurde 1998 mit dem Protokoll *XML-RPC* gelegt, das maßgeblich von Dave Winer in Zusammenarbeit mit Microsoft entwickelt wurde. XML besaß einen kleinen, kompakten Aufbau und diente dazu, Funktionsaufrufe per XML an einen Diensteanbieter zu senden. Dieser verarbeitete die Anfrage und schickt diese an die anfragende Applikation zurück. Im Gegensatz zum klassischen RPC verarbeitet diese Anfrage auf der Server-Seite nicht mehr ein Portmapper-Dämon erforderlich, sondern ein entsprechender XML-Prozessor.
Ein Beispiel einer XML-RPC-Anfrage könnte in etwa so aussehen. An einen Diensteanbieter wird eine XML-RPC-formatierte Anfrage geschickt:

```
<?xml version="1.0"?>
      <methodCall>
            <methodName>beispiel.statusabfrage</methodName>
            <params>
                  <param>
                   <value><i4>40</i4></value>
                  </param>
            </params>
      </methodCall>
```

In dieser Anfrage wird in einem Container namens *methodCall* eine Funktion namens "beispiel.statusabfrage" aufgerufen (definiert im Container *methodName*), die als Parameter die Zahl 40 überträgt. Das zurückgesendete Ergebnis vom Server könnte folgendermaßen aussehen:

```
<?xml version="1.0"?>
      <methodResponse>
            <params>
                  <param>
                   <value><string>Testantwort</string></value>
                  </param>
            </params>
      </methodResponse>
```

In diesem Fall wäre die Antwort also der Text "Testantwort".
Obwohl das zugegebenermaßen ein sehr einfacher Dialog ist, wird doch sehr schön deutlich, dass XML-RPC grundsätzlich einen recht einfachen und strukturierten Aufbau hat und vor allem keinerlei Programmcode oder ähnliches übertragen wird; ganz nach der Tradition von RPC werden lediglich Funktionen auf entfernten Systemen aufgerufen, Parameter übertragen und die Antworten entsprechend empfangen.

Aus <http://www.netplanet.org/remotedienste/rpc.shtml>

Dependency Injection

09 March 2015 15:06

If you have a class `Employee` and this employee has an `Address` you can have the `Employee` class defined as follows:

```
class Employee {
    private Address address;
// constructor
    public Employee( Address newAddress ) {
        this.address = newAddress;
    }
public Address getAddress() {
    return this.address;
    }
    public void setAddress( Address newAddress ) {
        this.address = newAddress;
    }
}
```

Everything looks fine so far.
This code shows a *HAS-A* relationship between the employee and his address, that's fine.
Now, this *HAS-A* relationship created a dependency between them. The problem comes within the constructor.
Each time you want to create an `Employee` instance you need an `Address` instance:

```
Address someAddress = ....
Employee oscar = new Employee( someAddress );
```

Working this way becomes problematic ***especially*** when you want to perform unit testing.
The main problem comes when you need to test one particular object, you need to create an instance of other object, and most likely you need to create an instance of yet **other** object to do that. The chain may become unmanageable.
To avoid this, you could change the constructor like this:

```
 public Employee(){
 }
```

Using a no args constructor.
Then you can set the address when ever you want:

```
Address someAddress = ....
Employee oscar = new Employee();
oscar.setAddress( someAddress );
```

Now, this may be a drag, if you have several attributes or if the objects are hard to create.
Yet, think about this, let's say, you add the `Department` attribute:

```
 class Employee {
     private Address address;
     private Department department;
....
```

If you have 300 employees, and all of them need to have the **same** department, and plus that same department has to be shared between some other objects (like the company list of departments, or the roles each department have etc) then you'll have a hard time with the visibility of the `Department`object and to share it through all the network of objects.

What the **Dependency Injection** is all about it to help you to, well, ***"inject"*** these dependencies in your code. Most of the frameworks allow you to do this by specifying in an external file, what object is to be injected.
Assume a properties file for a fictitious dependency injector:

```
  #mock employee
  employee.address = MockAddress.class
  employee.department = MockDepartment.class
#production setup
  employee.address = RealAddress.class
  employee.department = RealDepartment.class
```

You'll define what to inject for a given scenario.
What the Dependency Injector framework will do is to set the correct objects for you, so you don't have to code `setAddress` or `setDepartment`. This would be done either by reflection or by code generation or other techniques.

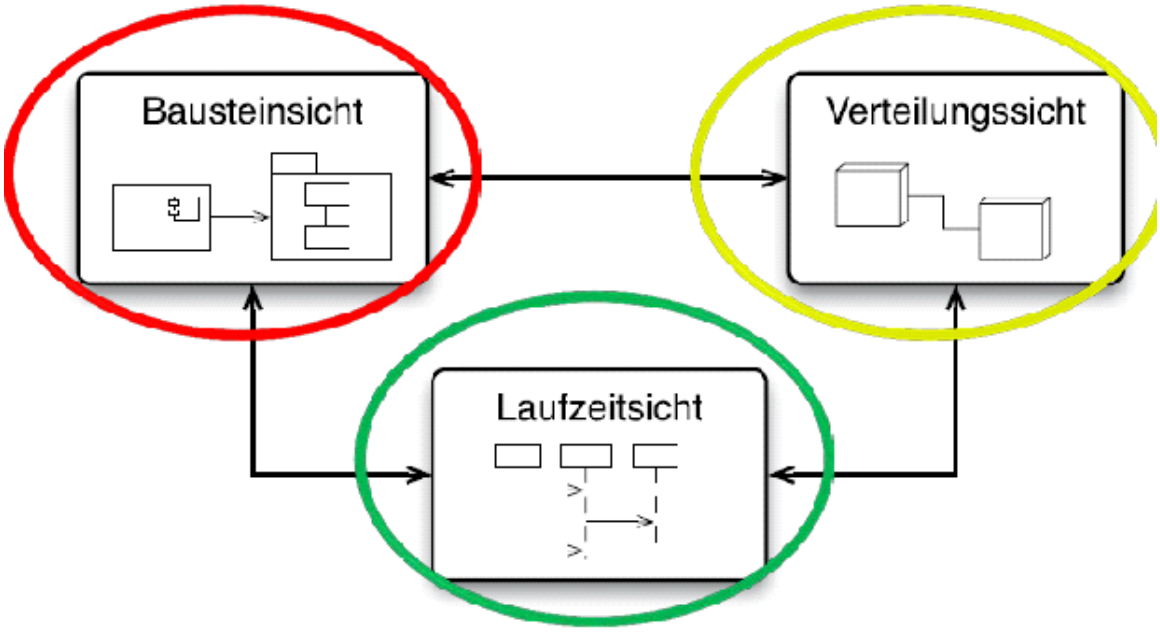
So, the next time you need to test the `Employee` class you may inject mock `Address` and `Departments` objects without having to code all the set/get for all your test. Even better, you can inject**real** `Address` and `Department` objects in production code, and still have the confidence your code works as tested.
That's pretty much about it.

Aus <http://stackoverflow.com/questions/1638919/how-to-explain-dependency-injection-to-a-5-year-old>

Sichten

09 March 2015 15:44

Für das Festhalten einer Software-Architetkur werden 3 Sichten empfohlen.



1. Bausteinsicht -> Seigt die statische Struktur des Systems
2. Laufzeitsich -> Zeigt das Verhalten des Systems
3. Verteilungssicht -> Zeigt die Verteilung von Systembestandteilen

Weitere Details Folien ab S. 50

Verteilte Systeme und Anwendung

12 March 2015 13:25

Entwicklung von Rechnersystemen

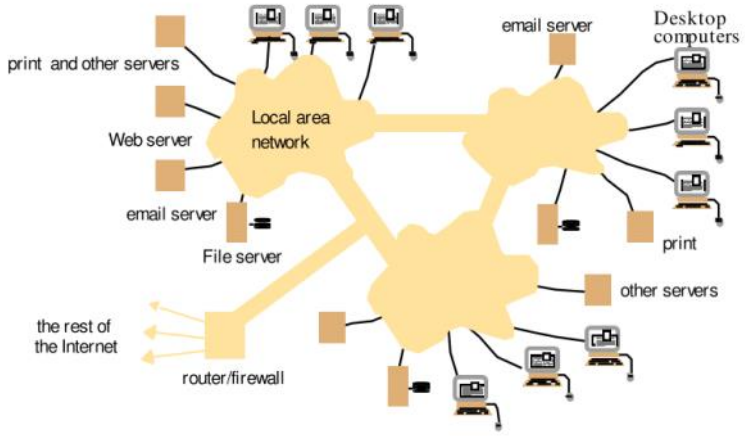
- 1960-1970
  - Grossrechner, Stapelverarbeitung
  - Terminals und erste Vernetzungen
  - Host-Host-Kommunikation
- 1970-1980
  - Grossrechner herrschend
  - Timesharing
  - TCP geboren (1978)
- 1980-1990
  - PC
  - PC-Netze
  - TCP/IP als Standard
- 1990-2000
  - WWW geboren 1990
    - Benutzt das Internet als verteiltes System
    - Basis für verteilte Anwendungen
  - Internet Boom
  - Vernetzung zunahme
  - Distributed Systems
  - Client-Server-Modell

Verteiltes System

besteht aus einer **Menge autonomer Computer**, die durch **Computernetzwerke** miteinander verbunden sind und mit einer **Software** für die **Koordination** ausgestattet sind

Ein verteiltes System ist ein System, in dem **Hard- und Softwarekomponenten**, die sich auf untereinander vernetzten Computer befinden, miteinander kommunizieren und Ihre Aktionen koordinieren, indem sie Nachrichten austauschen

Beispiel Intranet



Ist ein geschlossenes Netzwerk
Besteht aus Subsystemen

Vorteile

Besseres Abbild der Realität
- logische vs. physische Struktur
- Leistungen werden dort erbracht, wo sie benötigt werden

Wirtschaftlichkeit und Lastverteilung
- bessere Ausnutzung von Ressourcen
- Lastverbund

Bessere Skalierbarkeit
- Erweiterung von bestehenden Systemen nach Bedarf
- System funktioniert mit wenigen und auch mit Vielen Komponenten (Webserver & Browser im WWW-Kontext)

Fehlertoleranz
- Abfangen und Maskieren von Fehlern
- Lösungsansätze (HW- und SW-Redundanz, SW-Recovery)

Nachteile

Höhere Komplexität durch Verteilung und Heterogenität
- diverse Plattformen, HW-Architekturen, Programmiersprachen

Komplexe Netzinfrastruktur
- Netzwerkarchitekturen & Protokolle

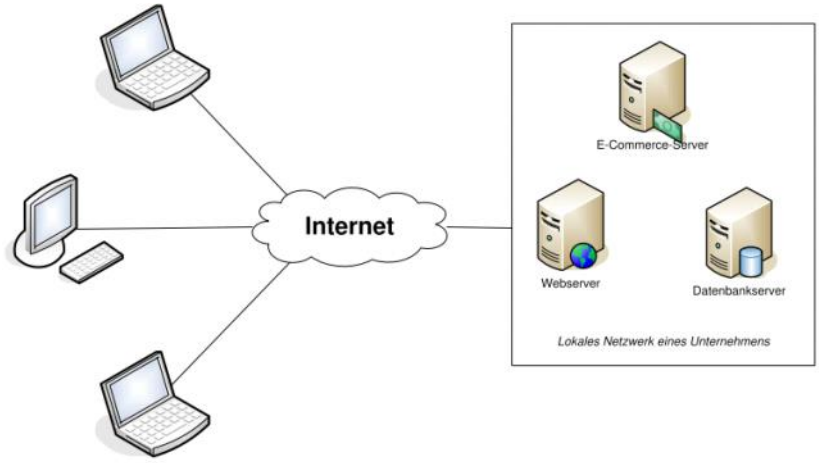
Höhere Sicherheitsrisiken (Verletzlichkeit)
- Vertraulichkeit / Integrität / Authentizität
- e-banking & e-commerce

Verteilte Anwendung

besteht aus verschiedenen Komponenten besteht und ein verteiltes System zur Lösung eines Anwendungsproblems benutzt.

Diese Komponeneten kommunizieren untereinander.

Beispiel Webanwendung



Besteht aus verschiedenen Schichten
- Model -> Datenhaltung

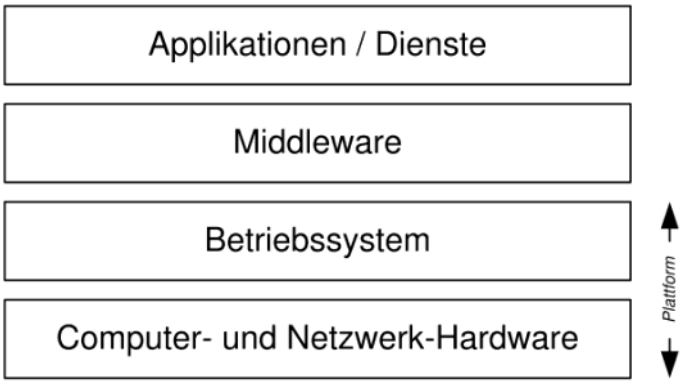
    - Datenbank
    - Dateien
- Controller -> Datenverarbeitung
    - Geschäftslogik
    - Prozesssteuerung
- View -> Datenpräsentation
    - Datensichten/ Views
    - Benutzerinterkation (UI/UX)

Modelle

12 March 2015 13:45

Bildet allgemeine Eigenschaften und Design eines Systems ab
- Komponenenten
- Interaktion zwischen Komponenten
- Verhalten der Komponenten

Softwareschichten



- Applikation
  - Unabhängig
  - Middleware-Modell
- Middleware
  - Übernimmt Rolle des Übersetzers
  - Verschleier Heterogenität des VS
  - Programmierungsmodell API
- Betriebssystem
  - Zugriff Resourcen

Architektur

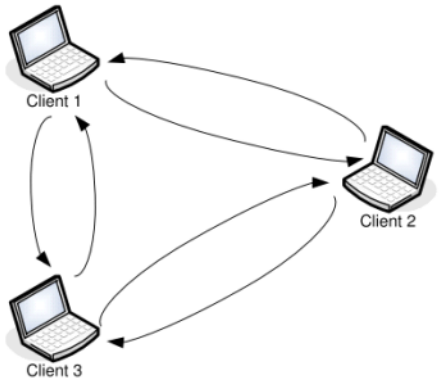
Vereinfach und abstrahiert die Erfassung von Funktionen einzelner Komponenten

- Synchrone Kommunikation
  - Hin und Her
- Asynchrone Kommunikation
  - Callback

Peer-ToPeer

Jeder Peer-Knoten hat
- Sowohl den Applikationscode
- als auch Koordinationscode

Anwendungsbereich
- Austauschplattform (Musik, Filme usw.)



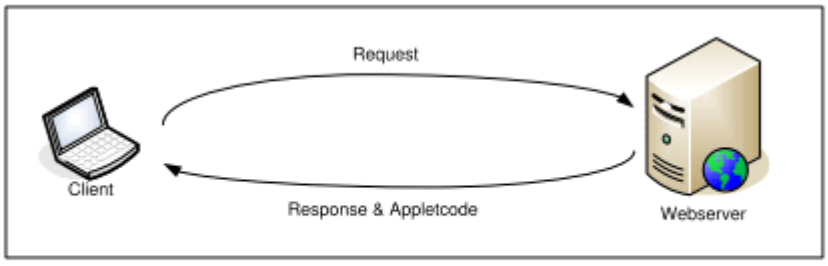
Client Server

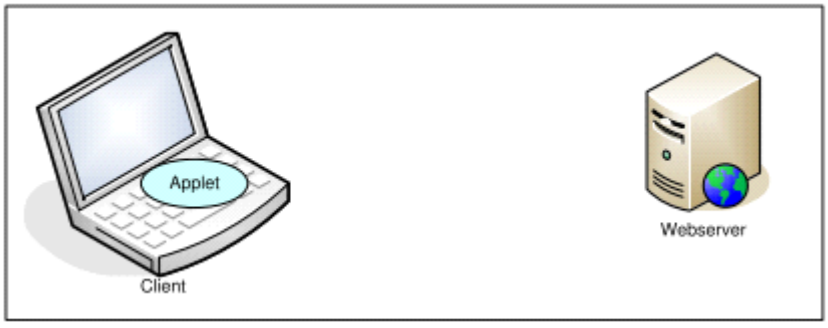
Besteht aus aktiver und passiver Komponente/Host
Kommuniziert über verschiedene Protokolle

- Http
- TCP/IP
- SMB
- FTP
- etc.

- Thin Client
  - Reine Präsentation von Daten
- Fat Client
  - Präsentation, Verarbeitung und teilweise Datenhaltung

Client Server Applet



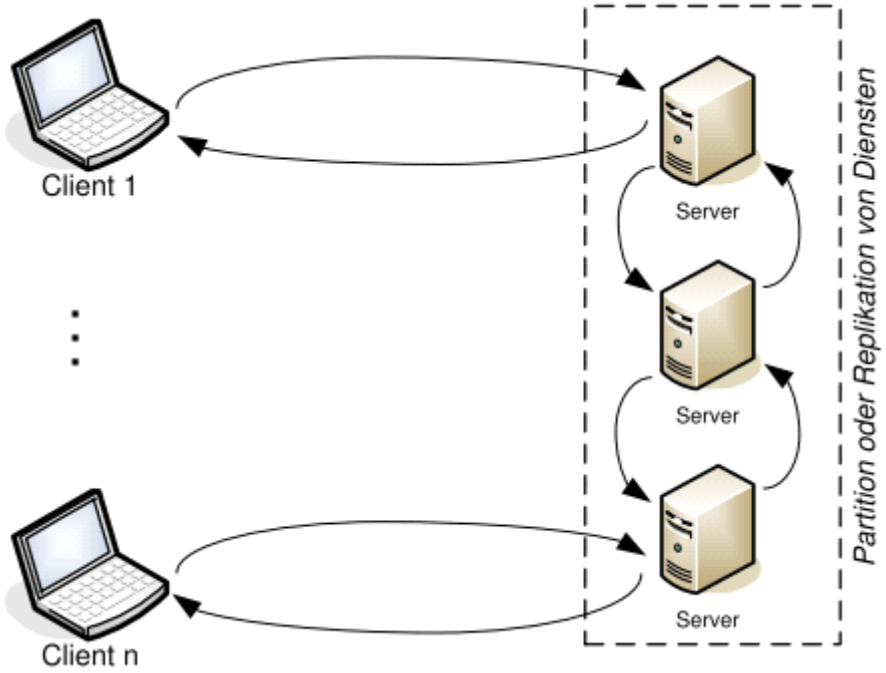


Applet wird auf Server gehostet, auf Client heruntergeladen und lokal gestartet.

Mehrfacher Server

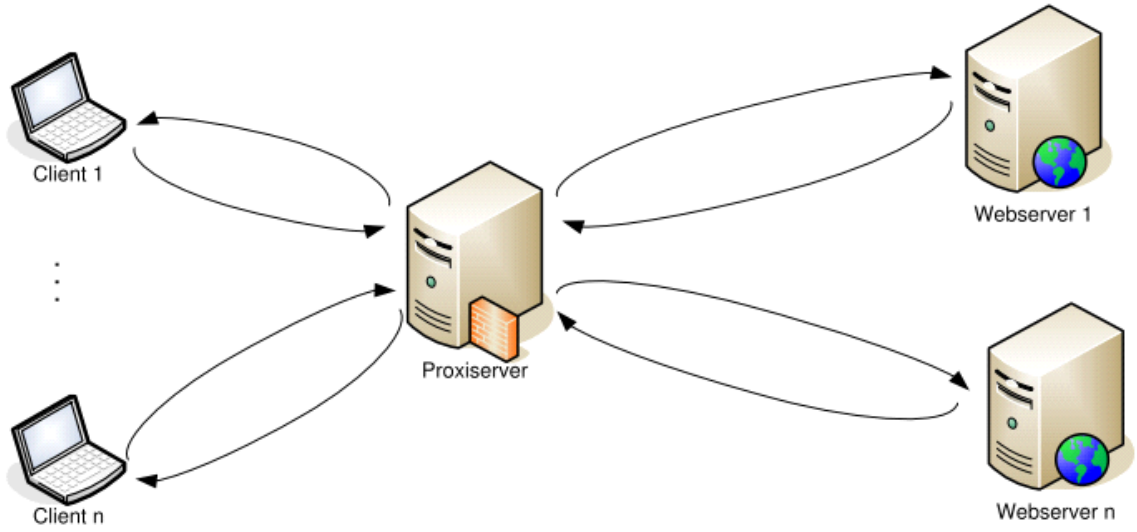
- Partition

    - Teilung von Platten
    - RAID
    - Im Netzwerk
        - CDN
        - Google Website
        - Cloudflare
- Replikation
    - Vermehren
    - Datenbackup



Proxy Server

- Erhöht Sicherheit
- Bessere Performance und Verfügbarkeit (Caching)
- Blendet Systeme aus



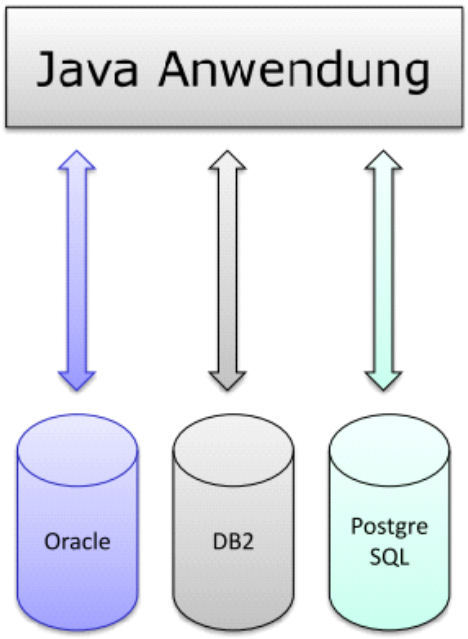
Anforderung Modellauswahl

- Performance
    - Anwortzeiten auf Anfragen

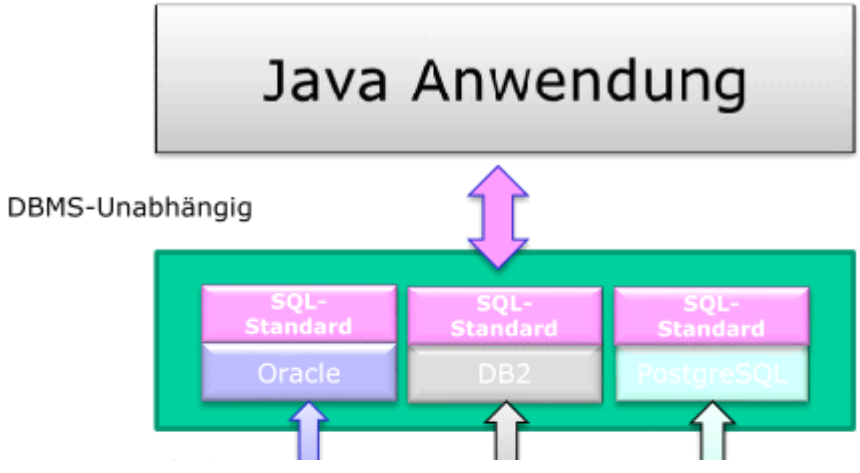
        - Wenig Komponenten
        - lokale Kommunikation
        - Kleine Datenmengen transferieren
    - Durchsatz von Daten
        - Durchsatz von Rechenarbeit
    - Lastbalancierung
        - Verteilt Arbeitslast auf verschiedene Kompoenten
- Dienstgüte (Quality of Service)
    - Cashing
        - Daten zwischen speichern
        - Problem: Aktualität der Daten
    - Replikation
        - Verfügbarkeit von Daten erhöht
        - Problem: Gleicher Stand
- Transparenz
    - VS wirkt wie eine Ganzeit
    - Benutzer nimmt Kompoententen des Systems nicht wahr
    - Typen:
        - Access Transparency (Zugriffstransp.)
        - Location Transparency (Ortstransp.) Concurrency Transparency (Nebenläufigkeittransp.)
        - Replication Transparency
        - Failure Transparency (Fehler- bzw. Ausfalltransp.)
        - Mobility Transparency (Mobilitätstransp.)
        - Performance Transparency
        - Scaling Transparency

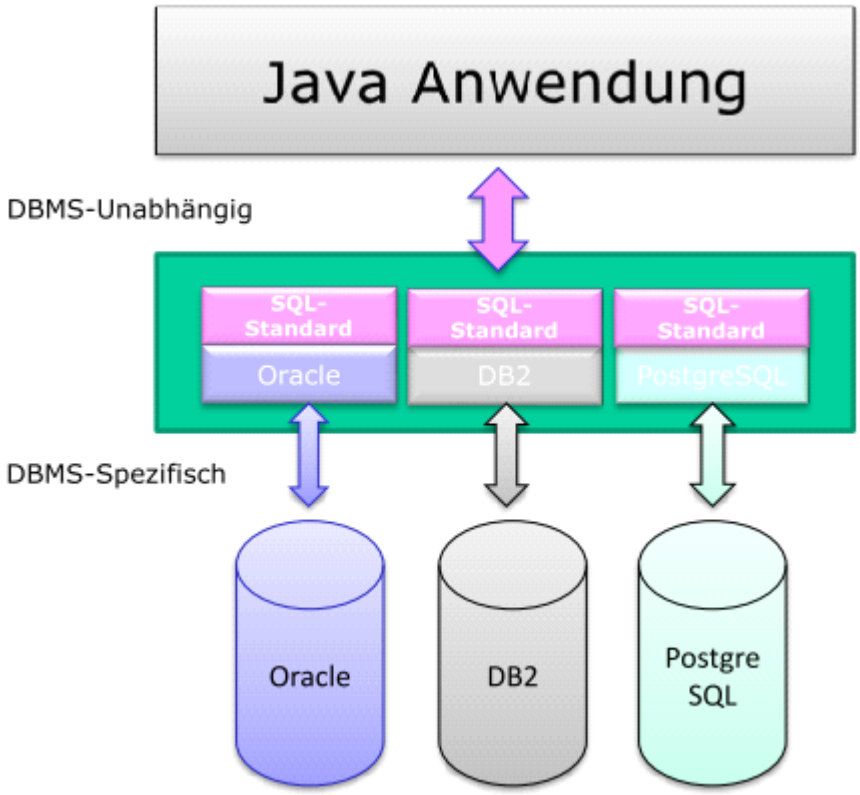
DBMS

12 March 2015 14:37

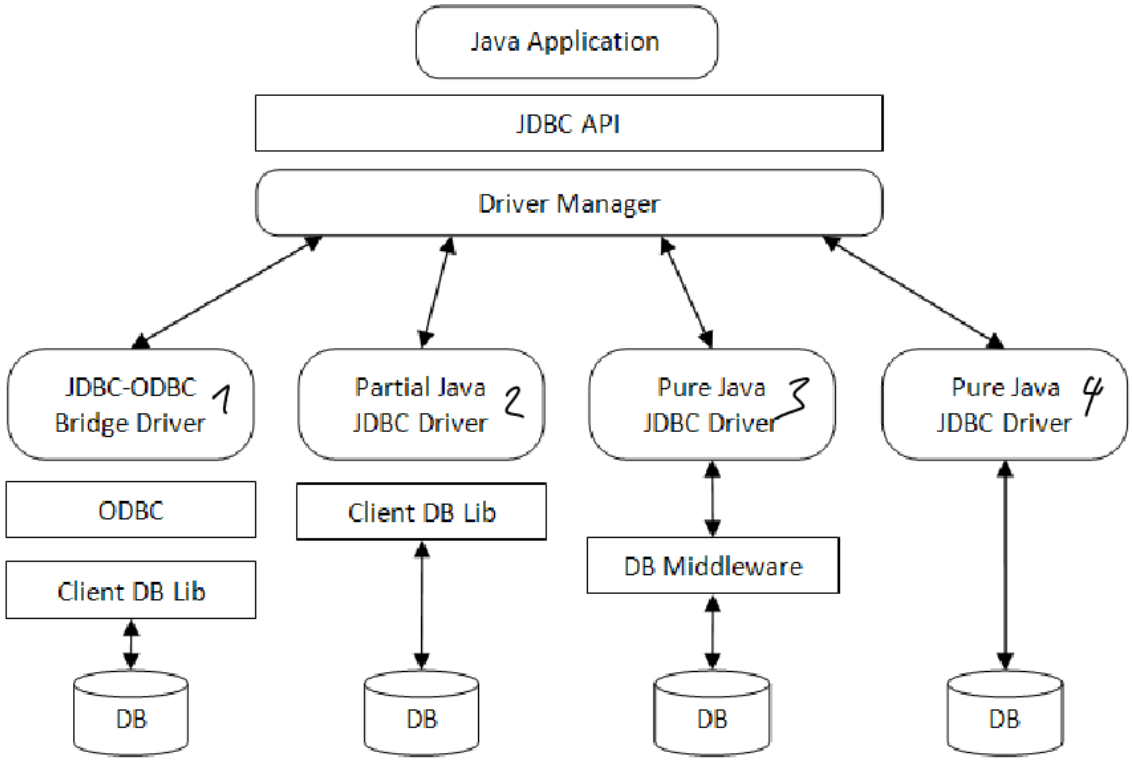


- Anwendung soll nicht abhängig vom DBMS sein.
- Zwischenschicht als Lösung
  - ODBC
    - Von Microsoft
    - Allgmeine Datenbank Zugriffsschnittstelle
  - JDBC
    - Robuster als ODBC
    - seit JDK 1.1 Bestandteil
    - Eigenes Programmierpaket
      - Treibermanager
        - verwaltet unterschiedliche DBMS Treiber
      - Treiber Test Suite
        - Testet die Treiber Anforderungen
      - JDBC-ODBC-Bridge
        - Übersetzt zu ODBC
      - API
        - Programmierschnittstellen
        - java.sql & javax-sql





JDBC-Treibertypen



Typ 1 und Typ 2:

- Übergangslösungen
- Enthalten Binärcode
- Nicht für Web-Anwendungen geeignet

Typ 3 und Typ 4:

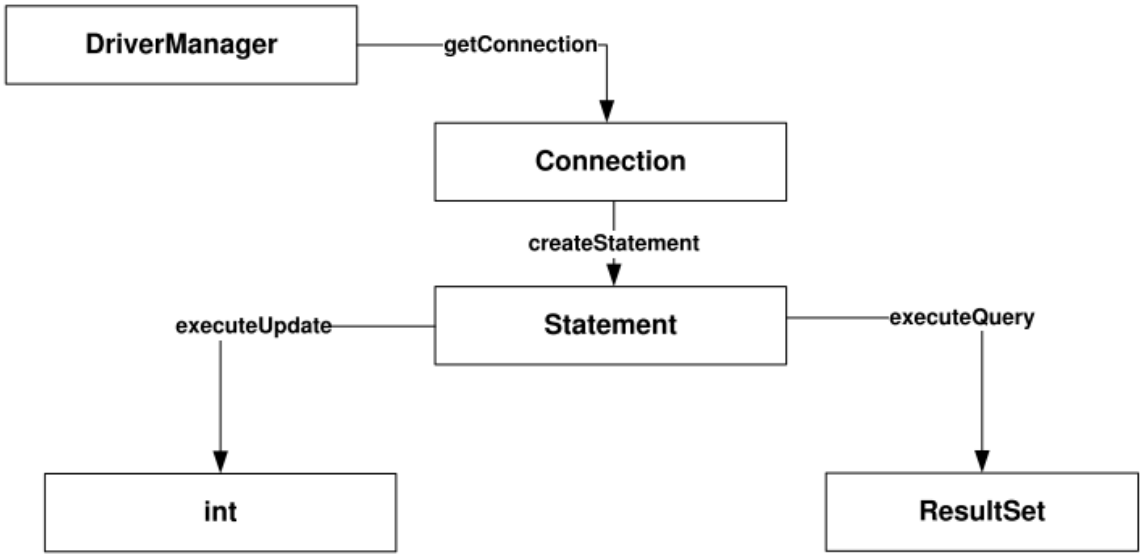
- Geeignet für die Web-Anwendungen
- In Java geschrieben

Typ 4:

- i.d.R. von DB-Hersteller geschrieben
- sehr effizient

Zugriff auf Datenbank

Klassen und Schnittstellen



- DriverManager verwaltet Treiber und baut Verbindung auf.
- ResultSet enthält abgefragte Daten
- Connection Objekt erlaubt Kommunikation mit Datenbank
- Statement ist eine SQL-Anweisung

Treiber laden

Laden der Treiberklasse

- Automatisch: jdbc.drivers werden beim Start der VM geladen -> wird selten gemacht.
- Explitzit: Über Methode forName der Klasse Class
  - PostgreSQL: Class.forName("org.postgresql.Driver")
  - MySQL: Class.forName("com.mysql.jdbc.Driver")

Erstellung Verbindung DB

Zugriff über Methoden DriverManager Class

- static Connection getConnection mit diversen Parameter
- Werden alle SQLException
- Gibt eine Connection-Objekt zurück
- Braucht URL:

    - PostgreSQL: jdbc:postgresql://localhost:5432/books_db
    - MySQL: jdbc:mysql://localhost/books_db
- Beispiel
- ```
  String url = "jdbc:postgresql://147.88.100.100:5432/raum_db";
  String user = "student";
  String pwd = "geheim";
  // Aufbau der Verbindung
  Connection con = DriverManager.getConnection(url, user, pwd);
  ```
- Wichtig: Verbindung testen und Ausnahmebehandlung machen

SQL Anweisung erstellen

Statement-Objekt mit Conection-Methode createStatement erstellen
Erzeugtes Objekt verwenden für
SELECT -> read
INSERT, DELETE, UPDATE -> lesend

Senden der Anweisungsfolge

- Über Statement-Methode executeUpdate ausführen
    - Liefert int-Wert zurück -> Anzahl
    - Beispiel:
    - ```
      int anz = 0;
      String delQuery =
      "DELETE FROM tbl_raum WHERE id_raum=2";
      anz = stm.executeUpdate(delQuery)
      ```
- Lesezugriff über executeQuery
    - Liefer Result-Set Objekt zurück
    - Beispiel:
    - ```
      String query = "SELECT * FROM tbl_raum";
      ResultSet rs = stm.executeQuery(query)
      ```

Verarbeiten der Ergebnisse

- Result-Set enthält Ergebnismenge der Abfrage
- Über methode next kann man schrittweise Tupel abfragen
- ResultSet kann auch direkt über generische get-Methoden abgefragt werden
- Beispiel:

```
String query = null;
ResultSet rs = null;
query = "SELECT bezeichnung, anz_plaetze FROM tbl_raum";
rs = stm.executeQuery(query);
```

- ```
  String str = null;
  while(rs.next()){
      str = "Raum: " + rs.getString("bezeichnung");
      str += ", Anz. Plaetze: " + rs.getInt("anz_plaetze");
      System.out.println(str);
  }
  ```

Schliessen der Verbindung

- Erfolgt über Methode close
    - Ist auch in diesen Schnittstellen enthalten
        - Connection
        - Statement
        - ResultSet
- Unterinstanzen werden automatisch geschlossen

Transaktionen

Erlaubt die Rücksetzung von nicht gelungenen Transaktionen

```
try {
    connection.setAutoCommit(false);

    statement.executeUpdate("INSERT ... ");
    statement.executeUpdate("UPDATE ... ");
    statement.executeUpdate("DELETE ... ");

    connection.commit();
}
catch (SQLException e) {
    if (connection != null){
       connection.rollback();
    }
}
```

Property-Datei

Auslagern von wechselenden Konfigurations-Parameter
Besonders interessante bei Konfiguration des DBMS
Beispiel:

```
// Properties-Objekt erzeugen
Properties dbProperties = new Properties();

// Klassenloader holen
ClassLoader cLoader = this.getClass().getClassLoader();

// Properties laden
dbProperties.load(cLoader.getResourceAsStream("db.properties");

// Treiber-Klasse auslesen
String driverClass = dbProperties.getProperty("jdbc.drivers");

// Treiber laden
Class.forName(driverClass);
```

Der Klassenlader holt einzelne Klassen in den Speicher.con

```
// URL, Benutzername und das Kennwort auslesen
String dbUrl = dbProperties.getProperty("jdbc.url");
String user = dbProperties.getProperty("jdbc.user");
String pwd = dbProperties.getProperty("jdbc.password");

// Verbindung zur Datenbank herstellen
con = DriverManager.getConnection(dbUrl, user, pwd);
```

Metadaten

Zusätzliche Informationen zum DBMS
JDBC bietet zwei Klassen

- DatabaseMetaData
    - DB Metadaten

- Zugriff über getMetaData
- Beispiel:

```
// Metadaten für die Datenbank holen
DatabaseMetaData dbMetaData = connection.getMetaData();
// Tabellennamen auslesen
ResultSet rSet = dbMetaData.getTables(null, null, null, new String[]{"TABLE"});
// Tabellennamen ausgeben
while(rSet.next())
{
     System.out.println("TABLE: " + rSet.getString("TABLE_NAME"));
}
```

- ResultSetMetaData
  - Result Metadaten
  - Zugriff über getMetaData
  - Beispiel:

```
// Metadaten des aktuellen ResultSet-Objekts holen
ResultSetMetaData rsMetaData = rSet.getMetaData();
// Anzahl Spalten auslesen
int anzahlSpalten = rsMetaData.getColumnCount();
// Spaltennamen und SQL-Typ ausgeben
if(rSet.next()) {
     for (int i = 1; i <= rsMetaData.getColumnCount(); i++) {
       System.out.println("COLUMN-NAME: " + rSet.getString(i) + "TYPE: " +
       rSet.getColumnTypeName());
     }
}
```

ORM

19 March 2015 13:19

Unterstützt das Speichern von Objekten in einem RDBMS in Form eines Frameworks.

Frameworks (Provider für JPA):

- Hibernate
- EclipseLink
- OpenJPA

JPA

Java Persistence API

JPA ist eine Spezifikation, mit der die für OR-Mapping benötigten Funktionalitäten in der Java-Welt definiert werden.

Vorteile:
- ermöglicht das Speichern von Daten auf OO-Weise (als Objekte)
- einfach im Gebrauch und nicht von irgendeinem Provider abhängig
- in der Industrie von führenden Unternehmen eingesetzt
- stellt die Portabilität von Anwendungen sicher
- kann sowohl mit JEE als JSE verwendet werden

Provider:
- Hibernate
- Toplink / EclipseLink
- OpenJPA
- MyBatis
- DataNucleus
- …

Aktuelle Version: JPA 2.0 (veröffentlicht in Dezember 2009)

Aufgaben

Verbindungsverwaltung
Mapping von Klassen und deren Felder auf Tabellen und Spalten (Attribute)
Formulierung und Ausführung von SQL-Befehlen
Transaktionssteuerung
Verwaltung von Relationen und Navigation darüber
Abbildung von Vererbungsbeziehungen
Generierung technischer Ids
Zurverfügungstellung objektorientierter Suchmöglichkeiten
Caching von Daten

Impedance Mismatch

```
public class Person{
    private String name;
    private String vorname;

    // Konstruktoren & Methoden …

}

public class Student extends Person {
    private String studium;
    private int semester;

    // Konstruktoren und Methoden …

}
```

Pro Vererbungshierarchie eine Tabelle

id	typ	name	vorname	studium	semester
...	...	...	...	...	...
27	**person**	Weber	Beat	null	null
28	**student**	Bucher	Roland	Turismus	5
29	**student**	Gugolz	Peter	Informatik	3
30	**person**	Kaumfmann	Kathrin	null	null
31	**student**	Lindauer	Daniel	Wirtschaft	4
...	...	...	...	...	...

- Vor- und Nachteile

Le Discriminator-Spalte

– viele Nullwerte

Pro konkrete Klasse

id	name	vorname
27	Weber	Beat
30	Kaumfmann	Kathrin
...	...	...

:Person

id	name	vorname	studium	semester
...	...	...	...	...
28	Bucher	Roland	Turismus	5
29	Gugolz	Peter	Informatik	3
31	Lindauer	Daniel	Wirtschaft	4
...	...	...	...	...

Pro konkrete und abstrakte Klasse der Hierarchie eine Tabelle

id_person	name	vorname
27	Weber	Beat
28	Bucher	Roland
29	Gugolz	Peter
30	Kaumfmann	Kathrin
31	Lindauer	Daniel
...	...	...

id_student	studium	semester
..	...	...
28	Turismus	5
29	Informatik	3
31	Theologie	4
...	...	...
...	...	...

:Person

:Student

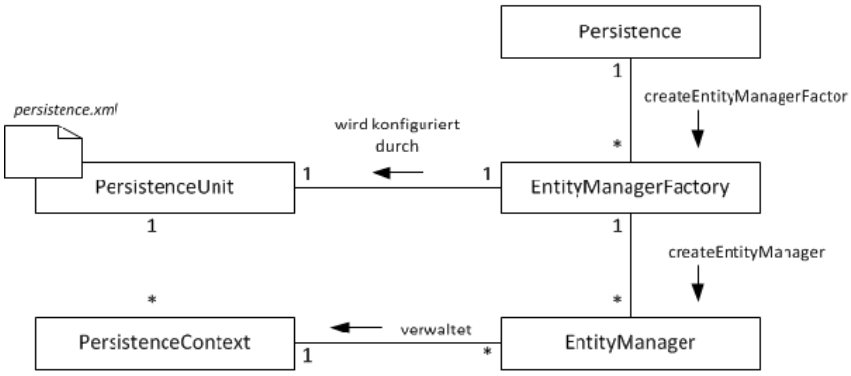
Wird meistens verwendet.
Schlüssel stimmen immer überein.

JPA Annotations

- Annnotation sind Sprachelemente die das einbinden von Metadaten in den Quellcode erlaubt.
- Beginnt immer mit @, dann Name und Parameter der Annoation.
- Werden vom Compiler verarbeitet.
- Beispiele
  - @Entity -> Im Normalfall der Klassenname kleingeschrieben.

- @ID -> PrimaryKey

JPA mit EclipseLink



Entity:
- Bezeichnung für leichtgewichtige Objekte (POJO), deren Zustände in einer Datenbank zu verwalten sind
- Beispiele: *Adresse*, *Person*, *Buch* ...

Klasse **Persistence** (*javax.persistence.Persistence*):
- für die Erzeugung der EntityManagerFactory-Instanz zuständig
- benötigt den Namen der "Persistence Unit"

Interface **EntityManagerFactory** (*javax.persistence.EntityManagerFactory)*:
- für das Erzeugen von EntityManager-Instanzen und
- den Aufbau der Verbindung zur Db zuständig

Interface **EntityManager** (*javax.persistence.EntityManager*):
- wird von der *EntityManagerFactory* kreiert und
- ist für die Verwaltung von Entities zuständig

Interface **EntityTransaction** (*javax.persistence.EntityTransaction*)
- ist für die Verwaltung von Transaktionen zuständig

Persistenzkontext:
- Die Summe aller von einem EntityManger verwalteten Entities
- Kann sich, je nach Bedarf, ändern:
  - neue Entities werden hinzugefügt
  - eine oder mehreren Entities werden entfernt (detached)

EntityManager muss zu jedem Zeitpunkt wissen, in welchem Zustand sich eine Entity, die er verwaltet (in seinem Kontext), befindet: ist sie neu hinzugefügt, aber noch nicht gespeichert worden, ist sie geändert worden (update) usw.

Erstellen der *EntityManagerFactory*
- kann (sinnvollerweise) in eine separate Util-Klasse ausgelagert werden
- die Klasse stellt eine Factory-Methode zur Verfügung, die eine *EntitiyManagerFactory* zurück liefert

Beispiel: Erzeugung einer EntityManagerFactory

```
public class JpaUtil {

    private static EntityManagerFactory entityManagerFactory = null;

    public static synchronized EntityManagerFactory getEntityManagerFactory() {

        if (entityManagerFactory == null) {
            entityManagerFactory = Persistence.createEntityManagerFactory("BooksPU");
        }

        return entityManagerFactory ;
    }
}
```

Kreieren eines EntityManager-Objekts:

```
EntityManagerFactor factory =
     JpaUtil.getEntityManagerFactory();

EntityManager em =
     factory.createEntityManager();
```

EntityManager Methoden:

```
void persist (Object entity)
<T> T find(Class<T>, Object primaryKey)
<T> T merge (T entity)
void remove (Object entity)
...
```

Transaktion "starten":

```
em.getTransaction().begin();
```

Objekte erstellen, Speichern …

```
Adresse adr = new Adresse ( … );

em.persist(adr);
```

Änderungen in die Db speichern:

```
em.getTransaction().commit();
```

EntityManager schliessen:

```
em.close();
```

Die Klasse, deren Instanzen als Entities in die Datenbank geschrieben werden sollen, muss mit entsprechenden Annotationen versehen werden

Beispiel:

```
@Entity
public class Adresse implements Serializable {
     @Id
     @GeneratedValue
     private int id;
     private String strasse;
     private int plz;
```

Assozationen

19 March 2015 13:46

Annahme Beziehung zwischen A und B

Besteht aus:

- Ownership: Entity A oder B
- Kardinialität: 1:1, 1:n, m:n
- Direktionalität: ja/nein
  - Unidirketional
  - Bidirektional
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany

Unidirektional - n:1

```
@Entity
public class Spieler implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    private String vorname;
    @ManyToOne
    private Team team;

    // Konstruktoren und Methoden …
}

@Entity
public class Team Spieler implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String name;

    // Weitere Attribute, Konstruktoren und Methoden …
}
```

Spalte heisst dann team_id im DMBS.

Soll Beziehung in zusätzlicher Verknüpfungstabelle realisiert werden gibt es @JoinTable.

```
@Entity
public class Spieler implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    private String vorname;
    @ManyToOne
    @JoinTable
    private Team team;

    // Konstruktoren und Methoden …
}
```

Resultat:

id	name
1	Team A
2	Team B
...	...

team_id	spieler_id
1	128
2	129
1	130
...	...

id	name	vorname
128	Weber	Marco
129	Meier	Johan
130	Ludwig	Marcel
...	...	...

Unidirektinal - 1:n

Andere Ansichtsweise dieser Beziehung.

```
@Entity
public class Person implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    private String vorname;
    @OneToMany
    private List<Email> emailAdressen;

    // Konstruktoren und Methoden …
}

@Entity
public class Email implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String emailAdresse;

    // Weitere Attribute, Konstruktoren und Methoden …
}
```

Auch hier kann man @JoinColumn verwenden.

Bidirektional - 1:n

Kombination der Unidirektionalen Beziehungen.

```java
@Entity
public class Verleger implements Serializable{
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    @OneToMany (mappedBy="verleger")
    private List<Buch> buchListe;

    // Konstruktoren und Methoden …
}

@Entity
public class Buch implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String titel;
    private String isbn;
    @ManyToOne
    private Verleger verleger;

    // Weitere Attribute, Konstruktoren und Methoden …
}
```

Unidirektional - 1:1

```java
@Entity
public class Person implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    private String vorname;
    @OneToOne
    private Personalausweis ausweis;

    // Konstruktoren und Methoden …
}

@Entity
public class Personalausweis implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String ausweisNummer;

    // Weitere Attribute, Konstruktoren und Methoden …
}
```

Person erhält personalausweis_id Spalte.
Kann mit @PrimaryKeyJoinColumn ersetzt werden.

Bidirektional - 1:1

```java
@Entity
public class Person implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    private String vorname;
    @OneToOne
    private Personalausweis ausweis;

    // Konstruktoren und Methoden …
}

@Entity
public class Personalausweis implements Serializable {
    @Id
    @GeneratedValue
    private Integer id;
    private String ausweisNummer;
    @OneToOne (mappedBy="ausweis")
    private Person person;

    // Weitere Attribute, Konstruktoren und Methoden …
}
```

Owner ist immer noch Person.
Mit @JoinColumn kann Name der Spalte beeinflusst werden.
Mit @JoinTable kann Name der Verknüpfungstabelle beeinflusst werden.

Unidirektional - n:m

```java
@Entity
public class Person implements Serializable {
    @Id
    @GeneratedValue
    private Integer int;
    private String name;
    private String vorname;
    @ManyToMany
    private List<Hobby> hobbyList;

    // Konstruktoren und Methoden …
}

@Entity
public class Hobby implements Serializable {
    @Id
    @GeneratedValue
    private Integer int;
    private String name;

    // Weitere Attribute, Konstruktoren und Methoden …
}
```

Dabei wird natürlich eine Verknüpfungstabelle eingerichtet.

id	name
1	Schwimmen
2	Wandern
3	Singen
4	Lesen
…	…

person_id	hobbyliste_id
128	1
129	1
130	2
129	3
129	4
130	4

id	name	vorname
128	Weber	Marco
129	Meier	Johan
130	Ludwig	Marcel
…	…	…

Bidirektional - n:m

```java
@Entity
public class Person implements Serializable {
    @Id
    @GeneratedValue
    private Integer int;
    private String name;
    private String vorname;
    @ManyToMany
    private List<Hobby> hobbyList;

    // Konstruktoren und Methoden …
}

@Entity
public class Hobby implements Serializable {
    @Id
    @GeneratedValue
    private Integer int;
    private String name;
    @ManyToMany (mappedBy="hobbyList")
    private List<Person> personListe;

    // Weitere Attribute, Konstruktoren und Methoden …
}
```

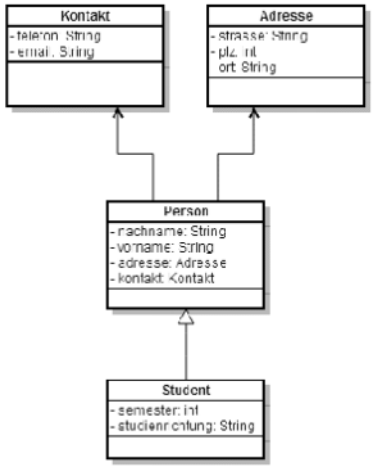
Objekte und Vererbung

19 March 2015 14:48

Die Speicherung von Objekten mit Vererbung muss speziell verarbeitet werden.

Beispiel:

```
public class Person{
    private String nachname;
    private String vorname;
    private Adresse adresse;
    private Kontakt kontakt;

    // Konstruktoren und Methoden …
}

public class Student extends Person{
    private int semester;
    private String studienrichtung;

    // Konstruktoren und Methoden …
}
```



Implementierung JPA:

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Person{
    @Id
    @GeneratedValue
    private int id;
    private String nachname;
    private String vorname;
    @OneToOne(cascade=CascadeType.ALL)
    private Adresse adresse;
    @OneToOne(cascade=CascadeType.ALL)
    private Kontakt kontakt;

    // Konstruktoren und Methoden …
}

@Entity
public class Student extends Person{
    private int semester;
    private String studienrichtung;

    // Konstruktoren und Methoden …
}
```

JOINED = 3. Variante

Ergebnis:

id	vorname	nachname	adresse_id	kontakt_id
...	...	...	...	...
16	Adrian	Portmann	6	9
19	Roland	Bucher	12	11
27	Daniel	Lindauer	14	16
...	...	...	...	...

Tabelle person

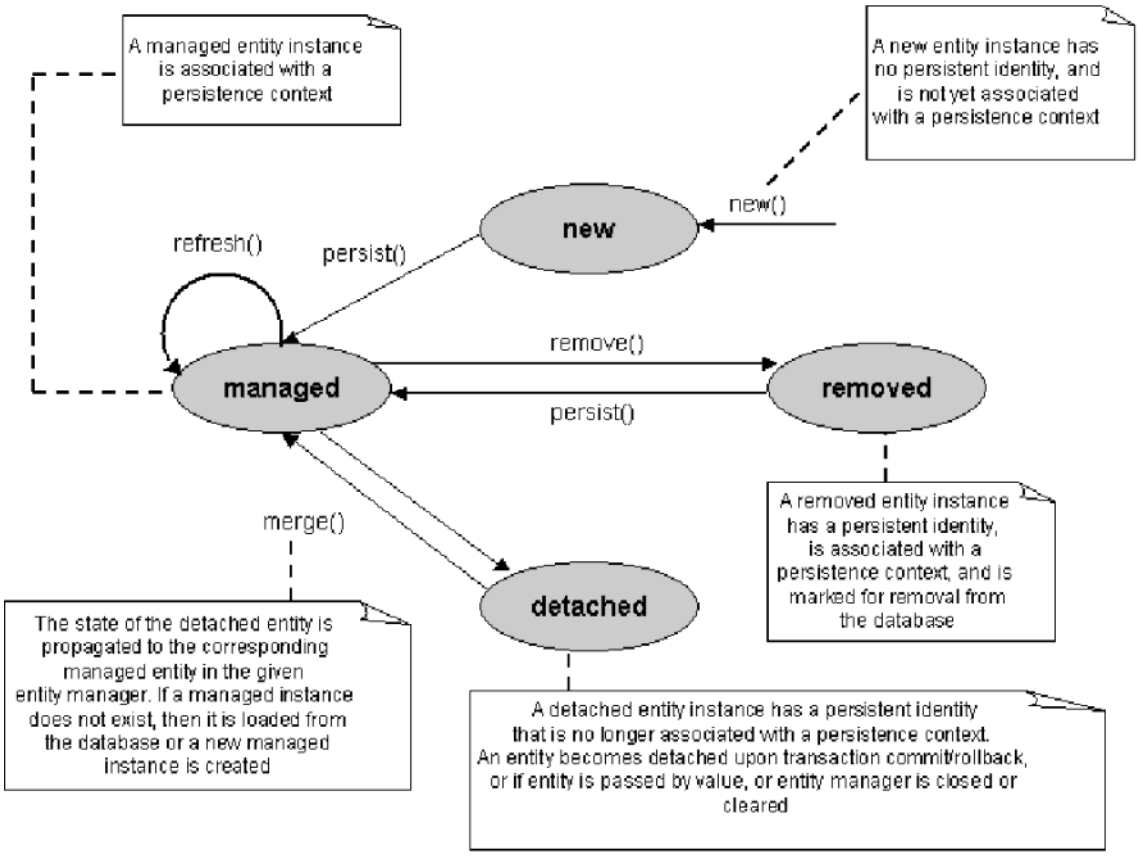
id	semester	studienrichtung
...	...	...
19	3	Informatik
...	...	...

Tabelle student

Lebenszyklus einer Entity

19 March 2015 14:54

Entity kann drei Zustände haben:



JPSQL

19 March 2015 15:16

Die zentrale Schnittstelle bei JPA-Abfragen

Immer dann gefragt, wenn der Entity-Primärschlüssel nicht bekannt ist

Eine *Query*-Instanz wird von dem *EntityManager* erstellt:

```
EntityManager em = JpaUtil.getEntityManagerFactory().createEnitityManager();

Query q = em.createQuery("SELECT p FROM Person p");
```

Die Abfrage kann mit einer der folgenden Methoden durchgeführt werden:

```
- List getResultList()
- Object getSingleResult()
```

Der Rückgabewert der beiden Methoden ist nicht typisiert und muss gecastet werden.

```
Person p = (Person) q.getSingleResult();
```

Ermöglicht JPA-Abfragen, die einen typisierten Rückgabewert zurückgeben

```
- List<X> getResultList()
- X getSingleResult()
- …
```

Eine TypedQuery-Instanz wird von einem *EntityManager* erzeugt:

```
EntityManager em = JpaUtil.getEntityManagerFactory().createEnitityManager();

TypedQuery<Person> tQuery = em.createQuery("SELECT p FROM Person p", Person.class);
```

Das Casten des Rückgabewertes nicht mehr nötig:

JPA ermöglicht das Einfügen von Parametern in eine konkrete Anfrage

Die Markierung von Parametern kann
- mit dem Namen oder
- mit dem Index

vorgenommen werden

Beispiele:

```
Query q = em.createTypedQuery("SELECT p FROM Person p where p.name=:n AND p.vorname=:v");
q.setParameter("n", "Pechvogel");
q.setParameter("v", "Hansli");
```

oder auch

```
TypedQuery<Person> q = em.createTypedQuery("SELECT p FROM Person p where p.name=?1 AND
p.vorname=?2");
q.setParameter(1, "Pechvogel");
q.setParameter(2, "Hansli");
```

Mit JPA ist es möglich, die benannten Abfragen zu definieren

Eine solche Abfrage wird mit einem eindeutigen Namen versehen und kann später beliebig oft verwendet werden

Definition:

```
@Entity
@NamedQuery(name="ch.hslu.wi.jpa.model.personByName" , query="SELECT p FROM Person p WHERE
p.name=:n");
public class Person {

    // Implementierung …
}
```

Verwendung:

```
Query q = em.createNamedQuery("ch.hslu.wi.jpa.model.personByName");
q.setParameter("n", "Pechvogel");
Person p = (Person) q.getSingleResult();
```

Vorteile:
- Wiederverwendung von Abfragen
- Erhöhte Wartbarkeit, da die Abfragen zentral verwaltet werden können

JPQL

Java **P**ersistence **Q**uery **L**anguage
- erinnert an SQL, ist jedoch im Gegensatz zu SQL voll objektorientiert
- ist nicht *case sensitive* – abgesehen von Klassen- und Attributnamen

Eine JPQL-Abfrage muss mindestens ein SELECT- und ein FROM-Block haben

Im FROM-Block wird
- der Wertebereich und
- die Identifikationsvariablen definiert, die in anderen Blöcken (z. B. WHERE-Block) verwendet (referenziert) werden können

Im SELECT-Block wird der konkrete Typ angegeben, der aus dem im FROM-Block definierten Wertebereich zurück geliefert werden soll

In einer JPQL-Abfrage können auch mehrere Entities angegeben werden

Solche Abfragen resultieren in einem Kartesischen Produkt aus den angegebenen Entities

Beispie:

```
SELECT p, a FROM Person p, Adresse a
```

Als Rückgabewert wird eine Liste von Objekt-Arrays zurückgeliefert, wobei in jedem Array je zwei Objekte zu finden sind: ein Person- und ein Adresse-Objekt

Vorsicht:
- Bei grossen Datenmengen kann eine solche Abfrage sehr aufwändig werden!

Die Einschränkung der Datenmenge kann mit Hilfe von WHERE vorgenommen werden, wobei mit unterschiedlichen Operatoren gearbeitet werden kann

Eine einfache Abfrage mit WHERE:

```
SELECT p FROM Person p WHERE p.vorname= 'Hansli'
```

Anfragen mit logischen und relationalen Operatoren

```
SELECT p FROM Person p WHERE p.name='Meier' AND p.vorname LIKE '%and'

SELECT a FROM Adresse a WHRE a.plz > 10000 OR ort LIKE '%ns'
```

Anfrage mit Menge-Operatoren

```
SELECT a FROM Adresse WHERE a.plz between '6000' AND '7000'
```

Anfrage mit Menge-Operatoren

```
SELECT a FROM Adresse WHERE a.plz between '6000' AND '7000'

SELECT a FROM Adresse WHERE a.plz IN ('6000', '6010', '6030', '6048')
```

JPQL erlaubt auch die Verwendung von String-Methoden

```
SELECT p FROM Person p WHERE p.vorname.length() > 5

SELECT substring(p.vorname, 2, 5) FROM Person p WHERE p.vorname='Hanspeter'
```

Das Sortieren der Ergebnismenge kann mit **order by** vorgenommen werden

```
SELECT p FROM Person p WHERE p.name='Meier' ORDER BY p.vorname DESC

SELECT a FROM Adresse a ORDER BY a.plz ASC
```

Native SQL

JPQL stellt noch viel mehr zur Verfügung:
- Aggregatfunktionen (*avg*, *sum*, *min*, *max*, *count*)
- Gruppierungsmöglichkeit mit *group by*
- Polymorphe Abfragen
- ...

Falls man einmal mit JPQL nicht weiter kommt, kann immer noch *native SQL* verwendet werden
- Das Query-Objekt wird von EntityManager mit Hilfe der Methode *createNativeQuery* erstellt
- der Methode wird eine gewöhnliche SQL-Abfrage als String übergeben

Beispiel (zum Veranschaulichen):

```
String sql = "SELECT * FROM adresse WHERE plz > 6000";
Query query = em.createNativeQuery(sql);
List objLst = q.getResultList();
…
```

oder auch

```
Query q = em.createNativeQuery(sql, Adresse.class);
List<Adresse> adrList = q.getResultList();
```

Fetching Strategien

JPA spezifiziert zwei Fetsching-Strategien: *Lazy Load* und *Eager Load*

Lazy Load:
- Daten werden aus der Db erst dann geladen, wenn sie benötigt werden.

Eager Load:
- Daten werden sofort vollständig geladen.

Die Angabe der Fetching-Strategie kann sowohl beim Attribut als auch bei der Beziehung angegeben werden

Beim Attribut:

```
@Basic (fetch = FetchType.LAZY)
private String meldung;
```

Bei der Beziehung

```
@OneToMany(fetch = FetchType.EAGER)
private List<Mitarbeiter> = new List<Mitarbeiter>();
```

Vorsicht bei der LazyLoad-Strategie:
- Spezifikation lässt zum Teil freien Raum
- Keine Pflicht für Provider

Bei Hibernate als JPA Provider:
- Das "Nachladen" kann nur dann durchgeführt werden, wenn sich die Entity immer noch im Persistenzkontext befindet bzw. von einer EntityManager-Instanz verwaltet wird
- Der Versuch, die Daten bei einer "detached"-Entity "nachzuladen", wird mit einer Exception quittiert

Bei EclipseLink als Provider
- Kann das "Nachladen" so lange realisiert werden, solange die Verbindung zu Db existiert
- Ein grosser Vorteil im Vergleich zu Hibernate!

Kaskadierung

Wenn ein komplexes Objekt z.B. gespeichert wird, müssen alle Teilobjekte einzeln gespeichert werden

Mit Kaskadieren lässt sich das "automatisieren":
- Wenn beispielsweise eine neue Person-Entity gespeichert wird, wird auch die zugehörige Adresse-Entity automatisch gespeichert
- Wenn eine neue Verleger-Entity gespeichert wird, werden auch alle Buch-Entities aus der *buchListe* des Verleger automatisch gespeichert
- Gleiches gilt auch für das Updaten und Löschen

Kaskadierung kann auf unterschiedliche Operationen angewendet werden, die Auswahl kann mit Hilfe des *CascadeType*-Enums angegeben werden:PERSIST, MERGE, REMOVE, REFRESH, DETACH und ALL

Beispiel A (Verleger):

```
@OneToMany (cascade = CascadeType.ALL)
private List<Buch> buchListe;
```

Beispiel B (Person):

```
@OneToOne (cascade = {CascadeType.PERSIST, CascadeType.REMOVE})
private Adresse adresse;
```

Im Beispiel A wird bei jeder Aktion, die auf einem Verleger ausgeführt wird, die gleiche Aktion auf allen Buch-Entities aus der *buchListe* auch ausgeführt
Im Beispiel B wird beim Speichern und Löschen einer Person-Entity auch die zugehörige Adresse-Entity auch gespeichert bzw. gelöscht

Nachtteile bie Kaskadierung:
- Kann evtl. die Performance negativ beeinflusst werden
- kann beim Löschen **sehr schnell viel Schaden** angerichtet werden!!!

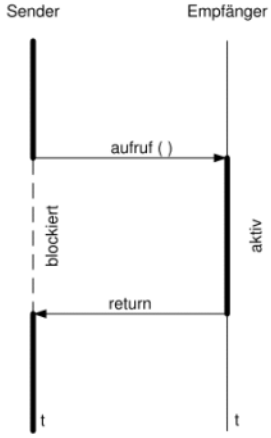
Kommunikationsmodelle

26 March 2015 13:23

Model legt Ablauf für die Kommunikatin zwischen Prozessen fest.
Die Kommunikation erfolgt entweder synchron oder asynchron.

Synchrone Kommunikation

- Wenn der Sender-Prozess eine Nachricht gesendet hat, muss er auf die Antwort warten.
- Wenn der Empfänger-Prozess ein "Empfangen" ausführt, wartet er so lange, bis eine Nachricht (eine Anfrage) empfangen wurde.
- Sehr verbreitet
  - Vorteile
    - Eher einfacher zum Implementieren
    - Synchronisation beim Zugriff auf gemeinsame Ressourcen gleich erledigt
  - Nachteile
    - zum Teil ineffizient (das Warten)
    - braucht sichere und schnelle Netzwerkverbindung
    - Empfängerprozess muss verfügbar sein
    - enge Kopplung zwischen Sender und Empfänger



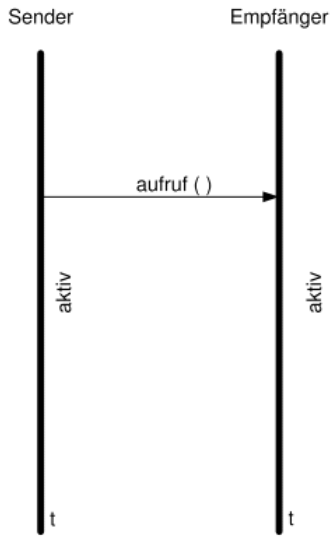
Synchrone Kommunikation

Asynchrone Kommunikation

- Der Sender-Prozess kann nach dem Senden der Nachricht sofort weiter arbeiten, ohne auf Antwort des Empfängers zu warten
- Der Empfänger-Prozess kann sowohl blockierend als auch nicht-blockierend Nachrichten empfangen
- Die Zustellung der Antwort:
  - Der Empfänger wird aktiv und stellt sie dem Sender bei Gelegenheit asynchron zu.
  - Der Sender holt sich bei Gelegenheit die Antwort selber vom Empfänger ab.
- Realisierung mittels Warteschlangen

- Vorteile
  - lose Kopplung von Prozessen
  - geringere Fehlerabhängigkeit
  - der Empfänger muss nicht empfangsbereit sein
  - effizienter (kein Warten)

- Nachteile
  - aufwendiger bzw. komplizierter in der Implementierung (Warteschlangen)
  - komplizierte Protokolle
  - nicht immer sinnvoll

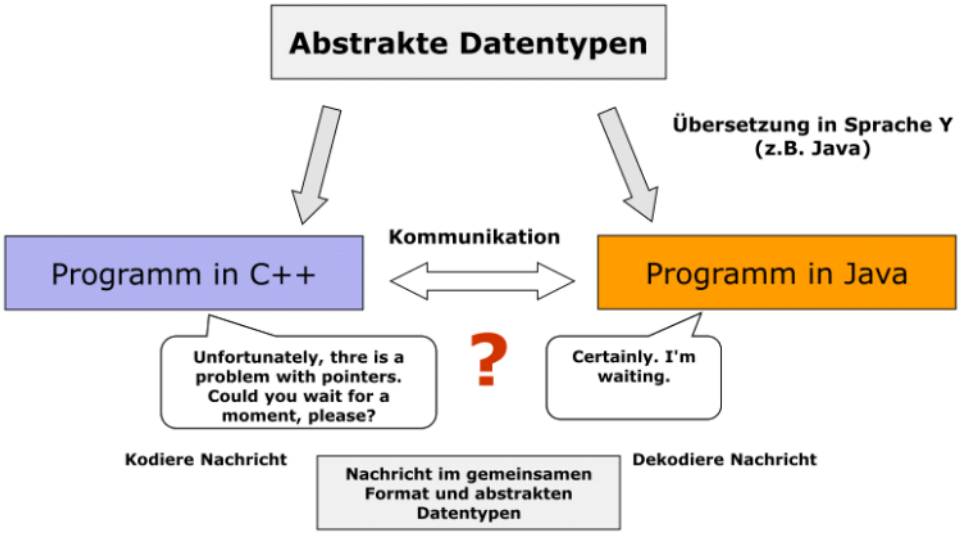


Asynchrone Kommunikation

Kommunikation

26 March 2015 13:27

Ein verteiltes System läuft in einer heterogenen Umgebung mit heterogenen Plattformen.
Dies führt zwangsweise zu Standardisierung der Kommunikationsskomponenten.

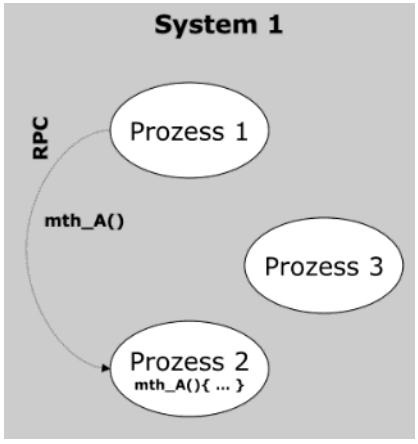


Es braucht verschiedene Werkzeuge um die Datentypen zu konvertieren.

LPC und RPC

Zwei Ansätze wie ein System auf unterschiedliche Prozesse zugreifen kann.

LPC

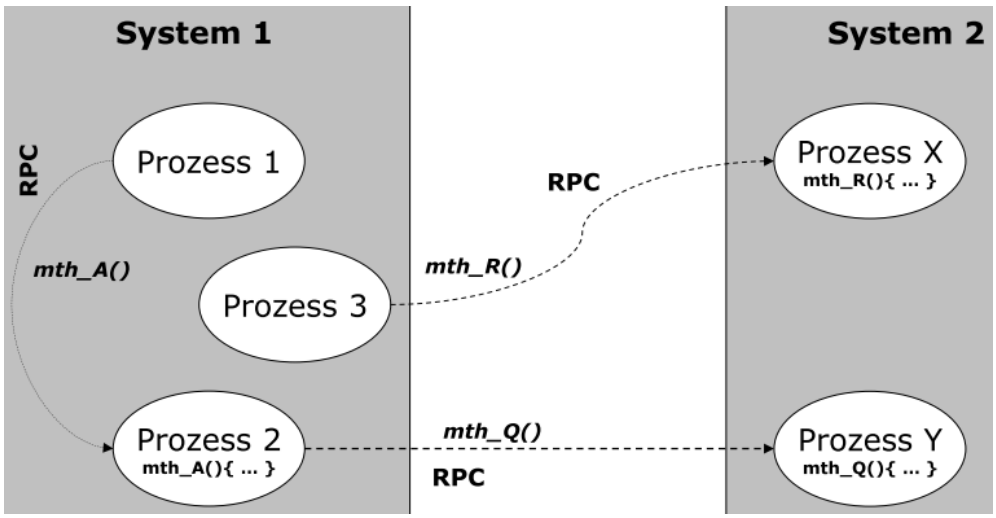


LPC steht für Local Procedure Call.

+ Perofrmance
+ Zuverlässig
+ Einfache Parameterübergabe
+ Einfacher Zugriff auf gemeinsame Daten
- Keine Interprozess kommunikation.

RPC

RPC steht für Remote Procedure Call.



+ Interprozesskommunikation
+ Aufruf von entfernten Prozeduren
- Performance
- Komplexität

Beide im Vergleich

- LPC ist einfacher zu implementieren, da sich die ganze Kommunikation (Nachrichtenaustausch) in einem gleichen Speicherraum (gleicher Prozess) abspielt

- RPC ist weniger performant
  - Kommunikation erfolgt in Form von Nachrichten, welche von einem zu anderem Prozess übertragen werden
  - Parameter, welche übergeben werden müssen, müssen in die Nachricht passend verpackt werden
- RPC ist komplexer in der Implementierung, da zusätzliche Protokolle für das Senden von Nachrichten benötigt

Interprozesskommunikation

Zur Kommunikation unterscheidet sich die unterteilung in Anwort und Anfrage.

- Request
  - Anfrage Nachricht
  - Ruft entferne Operation auf
  - Übergibt Parameter

- Reply
  - Gibt Anwort auf Request
  - Enthält Rückgabewert der Operation
  - Wirft Ausnahmen

Beispiel:

Entfernte Operation:

```
public boolean login (String username, String password) throws Exception
```

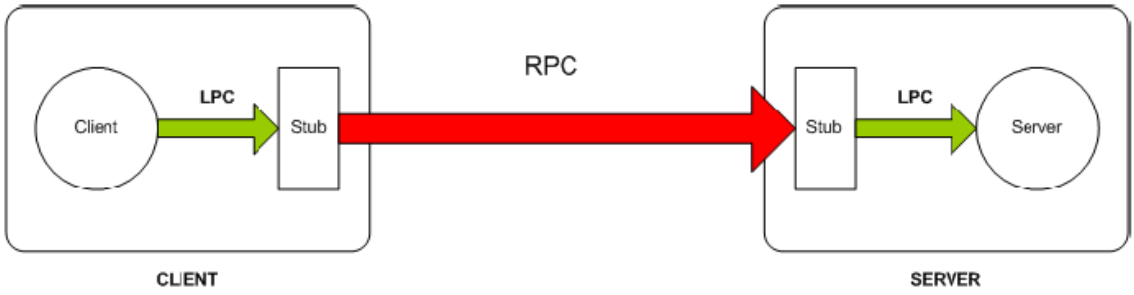
Request-Protokoll:  methode # username # password

Reply-Protokoll:  status # true | false # null | Exception

RPC

26 March 2015 13:45

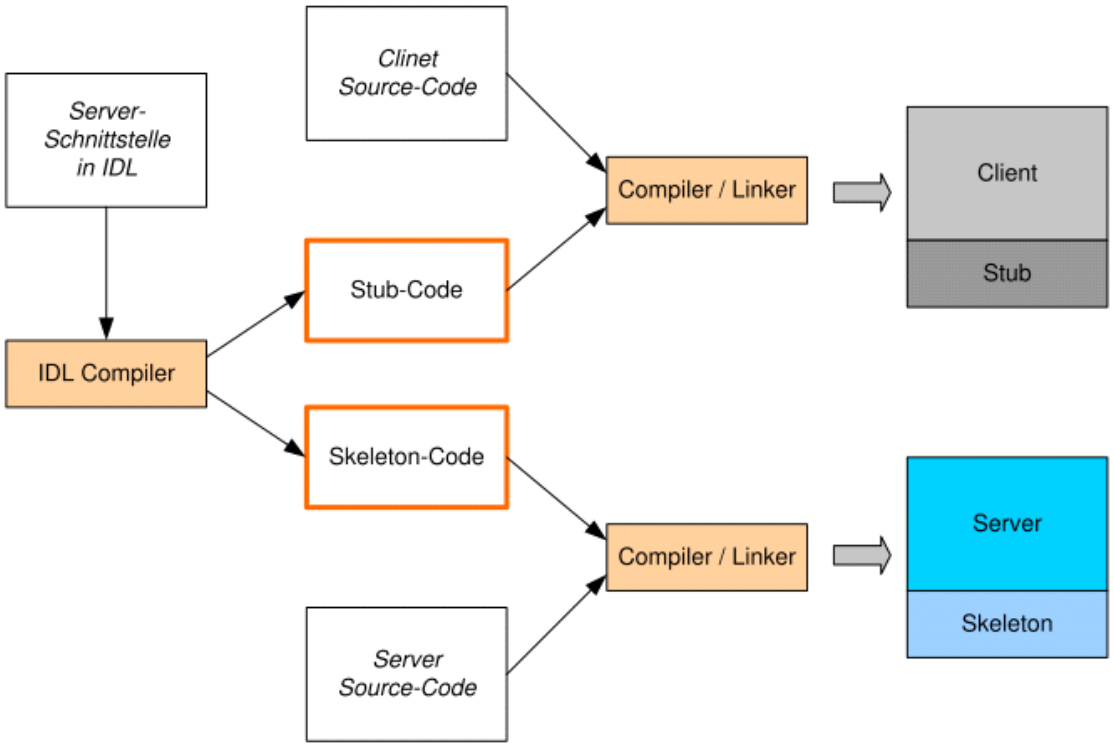
Im gleichen Prozess stellt man eine Komponente zur Verfügung, die wie ein Server aussieht.
Diese Komponente kopiert die Schnittstellen des realen Servers.
Man spricht hier von Stub.
Dasselbe passiert auf dem Server.
Man spricht hier von ServerStub.



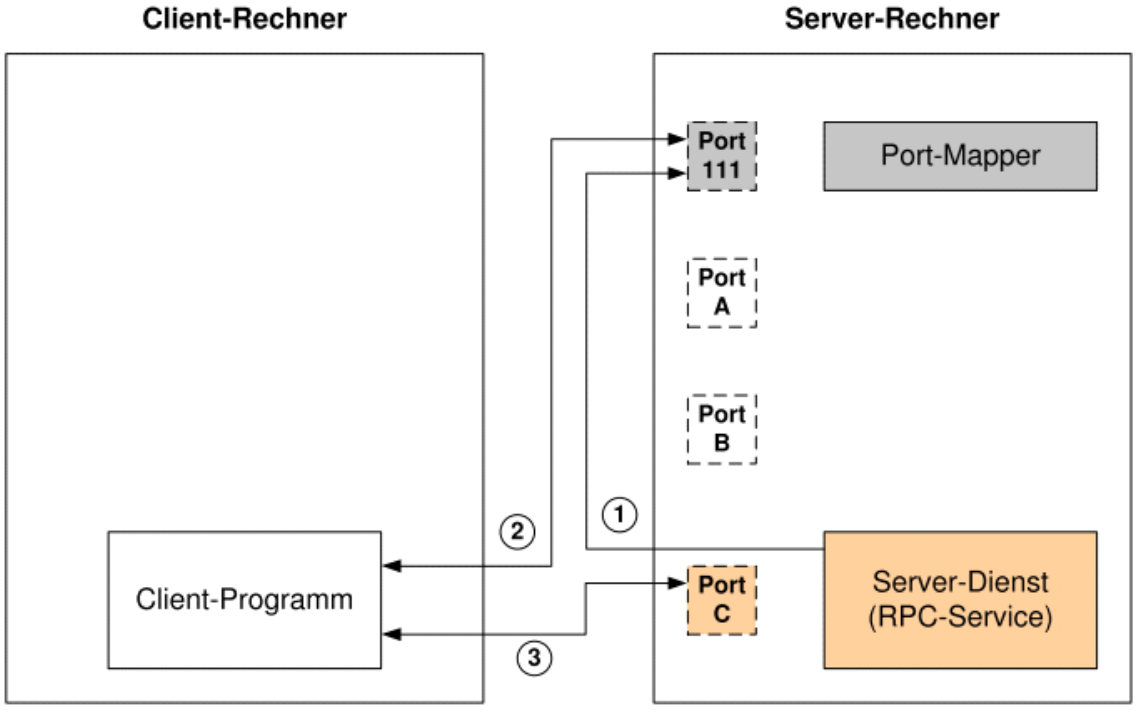
Das ganze läuft dann wie folgt ab:

1. Client ruft die Server-Methode auf dem ClientStub auf
2. ClientStub verpackt den Aufruf in eine übertragbare Nachricht
3. Die Nachricht wird zum Server durch die Netzwerk-Runtime gesendet
4. Empfangene Nachricht wird zum ServerStub weiter geleitet
5. ServerStub entpackt die Nachricht und ruft die Server-Prozedur auf
6. Die aufgerufene Server-Prozedur liefert die Antwort zurück
7. ServerStub verpackt die Antwort in eine übertragbare Nachricht
8. Die Nachricht wird zum Client durch die Netzwerk-Runtime gesendet
9. Netzwerk-Runtime empfängt die Nachricht und leitet sie an den ClientStub weiter
10. ClientStub entpackt die Antwort und leitet sie an den Client weiter

Probleme von RPC



-Keine Built-In unterstützung
-Precompiler für Stubs
-Definition von Schnittstellen über IDL (Interface Definition Language)
-Unterschiedliche Plattformen
Verschiedene Standards wurden geschaffen.
-Kommunikationspartner muss gefunden werden.
Binding Dienst Verwaltet Services und Potnummern
Bei Sun RPC: portmapper



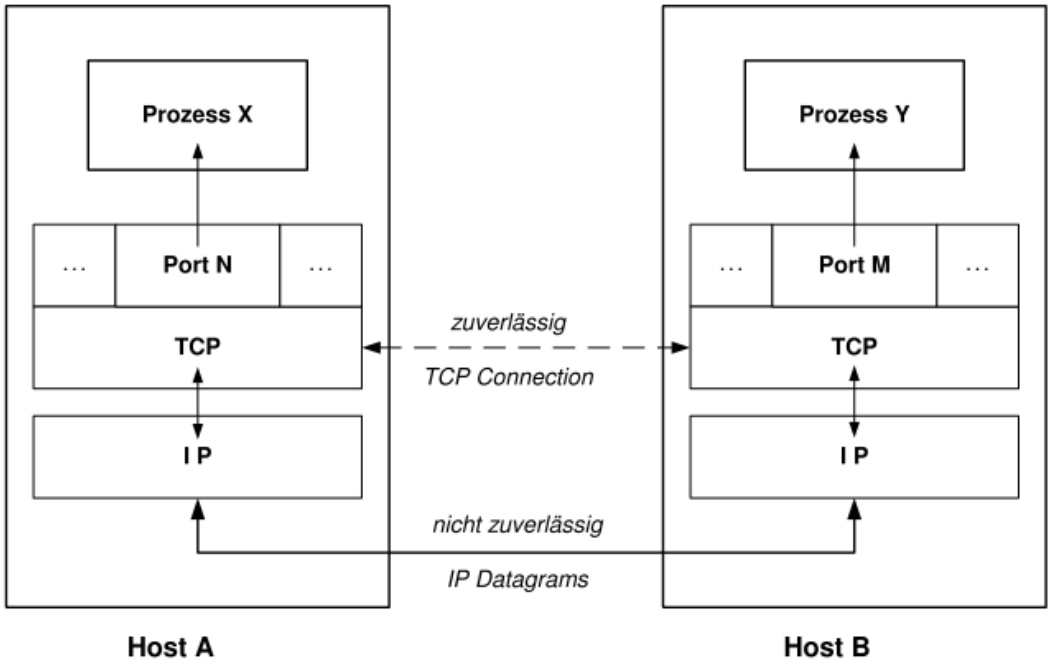
1. Server start Service und registiert bei Binding-Dienst
2. Client frag beim Binder nach Portnummer
3. Client verbindet sich mit Service mit erhaltener Portnummer.

Socket

26 March 2015 14:13

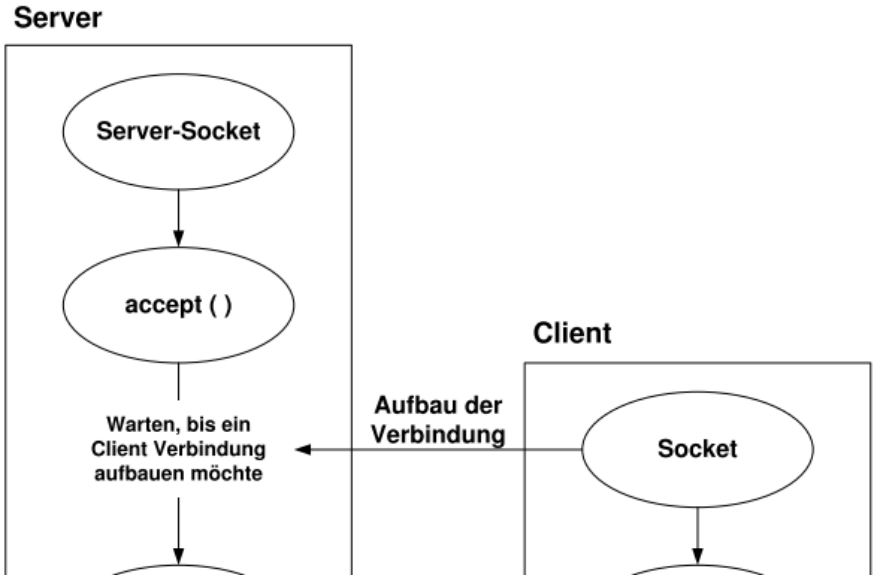
Socket verbindet Kommunikationsendpunkte.
Es gibt zwei Arten
- Streamsocket (TPC)
- Datagramsocket (UDP)

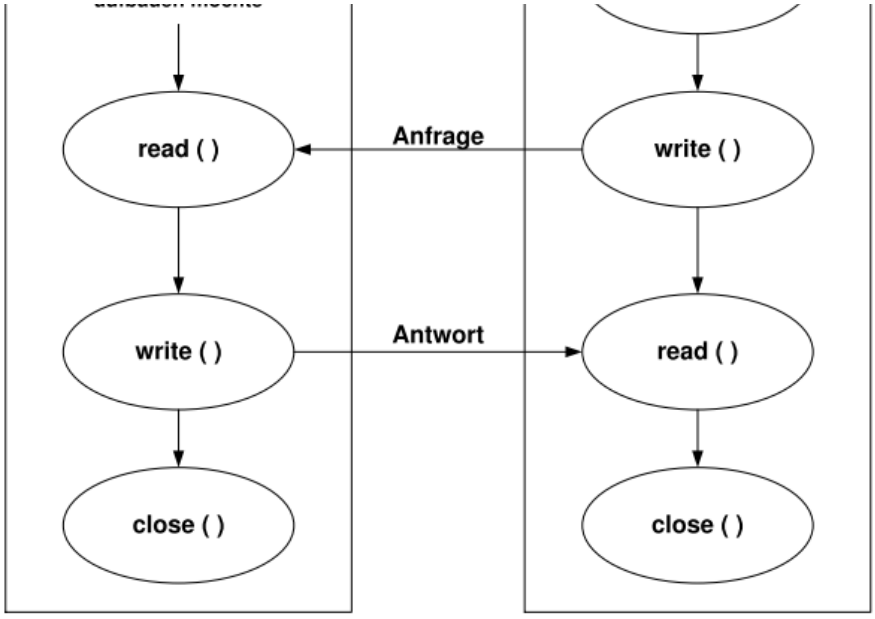
Braucht IP und Port-Nummer zum verbinden.
Rechner kann mehrer Sockets haben.



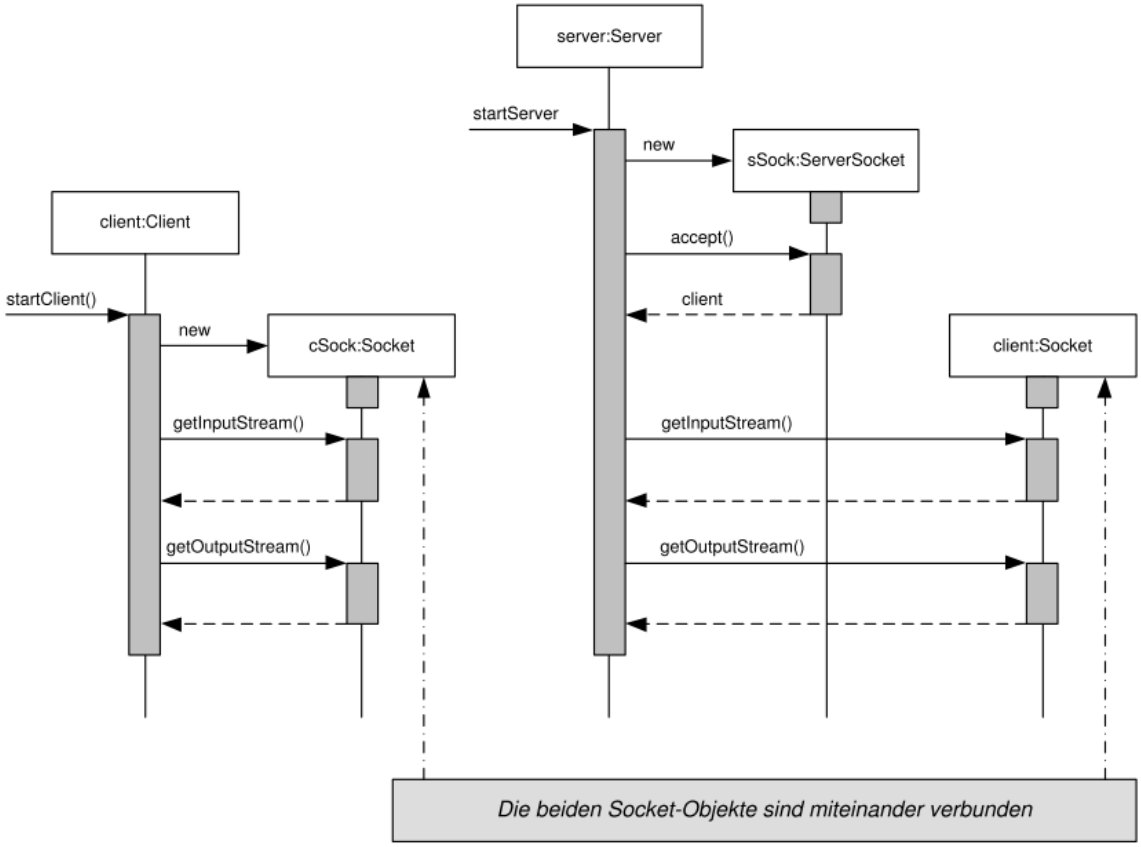
TCP-Sockets

Verbindungsorientierte Kommunikation
Phasen:
- Aufbau Verbindung
- Übertragen Daten
- Abbau Verbindung





Beide Kommunikationspartner bauen einen Socket auf.
:Dazu das Sequenzdiagramm



TCP-Sockets in Java

Enthalten in java.net.ServerSocket und java.net.Socket (client)

Beispiel - Server:

```
// ServerSocket erzeugen
ServerSocket server = new ServerSocket(port);

// ClientSocket holen, wenn eine Verbindung gewuenscht wird
Socket client = server.accept();

// Informationen ueber den Client ausgeben
String hostName = client.getInetAddress().getHostName();
int p = client.getPort();
System.out.println("Verbindung mit: " + hostName + ", Port: " + p + "\n");

// InputStream vom Client holen
InputStream is = client.getInputStream();

// vom Client zugestellten Daten ausgeben
int c = 0;

while ((c = is.read()) != -1)
{
    System.out.print((char) c);
}
```

Beispiel - Client:

```
// Socket erzeugen und Verbindung zum Server aufbauen
Socket socket = new Socket("localhost", 10001);

// Benachrichtigung
System.out.println("Verbindung mit Server hergestellt!");

// OutputStream vom Socket holen
OutputStream os = socket.getOutputStream();

// Meldung dem Server senden
String msg = "Das ist eine Test-Meldung!";
os.write(msg.getBytes());

// Verbindung schliessen
socket.close();
```

UDP-Sockets in Java

Sind nicht verbindungsorientiert -> Keine Kontrolle zu Datenverlust.
Daten werden in max. 8 KB grosse Pakete geschrieben (Wenn grösser sgementierung)
Enthalten in java.net.DatagramSocket (Listener) und java.net.DatagramPacket (Sender)

Beispiel - Server:

```
DatagramSocket socket = null;
DatagramPacket req = null, res = null;
byte[] buf = new byte[508];

// UDP-Socket erzeugen
socket = new DatagramSocket(9001);

// Request-DatagramPacket erzeugen
req = new DatagramPacket(buf, buf.length);

while(true)
{
    // Request empfangen (entgegennehmen)
    socket.receive(req);
    // Response erzeugen
    res = new DatagramPacket(req.getData(), req.getLength(),
                             req.getAddress(), req.getPort());
    // Response senden
    socket.send(res);
}
```

Beispiel - Client:

```
String msg = "Das ist eine Test-Meldung!";
byte [] resBuf = new byte[508];
int port = 9001;
InetAddress server = InetAddress.getByName("localhost");

// Socket erzeugen
DatagramSocket socket = new DatagramSocket();

// Request erzeugen
DatagramPacket req = new DatagramPacket(msg.getBytes(), msg.length(),
                                        server, port);
// Request senden
socket.send(req);

DatagramPacket res = new DatagramPacket(resBuf, resBuf.length);

// Response empfangen
socket.receive(res);

System.out.println("Antwort: " + new String(res.getData()));
```

Multicast Sockets in Java

Java kennt zusätzlich noch Multicast Sockets.
Die gleiche Nachricht wird an mehre Partner gesendet.
Enthalten in java.net.MulticastSocket.

Beispiel - Sender:

```
String msg = "Diese Meldung geht an alle Gruppenmitglieder!";
MulticastSocket s = null;
DatagramPacket mOut = null;
String mcIp= "230.2.2.2";

InetAddress group = InetAddress.getByName(mcIp);
// Socker erzeugen
s = new MulticastSocket(4004);
// Der Gruppe beitretten
s.joinGroup(group);
// Nachricht erzeugen
mOut = new DatagramPacket(msg.getBytes(), msg.length(), group, 4004);
// Nachricht senden
s.send(mOut);
// Gruppe verlassen
s.leaveGroup(group);

if (s != null) s.close();
```

Beispiel - Receiver:

```
MulticastSocket s = null;
DatagramPacket mIn = null;
String mcIp= "230.2.2.2";

InetAddress group = InetAddress.getByName(mcIp);
// Socker erzeugen
s = new MulticastSocket(4004);
// Der Gruppe beitretten
s.joinGroup(group);

mIn = new DatagramPacket(buf, buf.length, group, 4004);

// Nachricht empfangen
s.receive(mIn);
// Nachricht ausgeben
System.out.println("Empfangen: " + new String(mIn.getData()));
// Gruppe verlassen
s.leaveGroup(group);

if (s != null) s.close();
```

Zusammenfassung

Vorteile

- sehr gute Performance
- Protokolle können frei nach Bedarf implementiert werden, wodurch Anpassungen bzw. Optimierungen möglich sind

Nachteile

- Sockets eignen sich für reine Datenübertragung
- Sockets sind nur die Schnittstelle zur Transportschicht
- Bieten keine Unterstützung für Kommunikationssteuerung und Datendarstellung
- Protokolle müssen auf der Anwendungsebene implementiert werden um Kommunikationspartner zu finden und Austausch von Nachrichten zu steuern
- Eigenschaften der Transportschicht sind zu berücksichtigen (TCP/ UDP)
- Programmierung relativ aufwendig und fehleranfällig

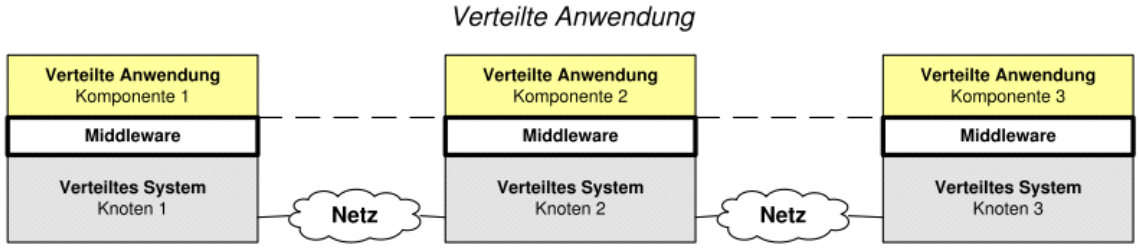
Middleware

26 March 2015 13:18

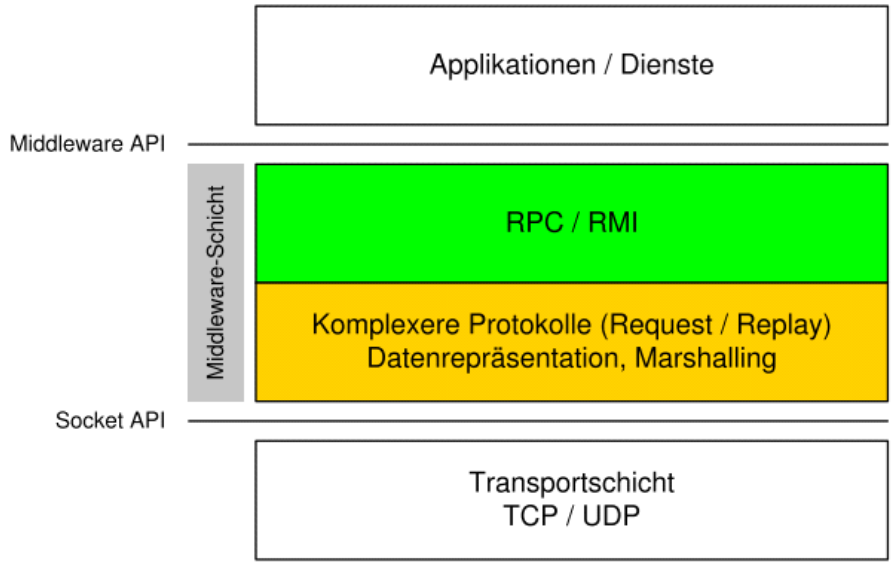
Vermittelt zwischen verteilten Anwendung und der unterliegenden Schicht

- Interaktion zwischen Komponenten vereinfachen
- Komplexität maskieren

Horizontale Architektur



Vertikale Architektur



Aufgaben

Abschrimung Programmierer/ Nutzer von komplexen Netzwerkprotokollen und Strukturen
Ortstranzaprenz: Client erkennt nicht wo sich Objekt befindet
Verbergen der Transportprotokolle
Plattform unabhängig
Abstraktion der Netzwerkprogrammierung

Kommunikations Orientierte Middleware (KOM)

Konzentriert sich auf die Kommunikationsinfrastruktur für Komponenten von verteilten Anwendungen.

Aufgaben

- Kommunikation
- Marshalling und Unmarshalling (Netzerkpakete ver- und entpacken)
- Fehlerbehandlung
  - Übtragungsfehler
  - Ausfall von Komponenten

Anwendungsorientierte Middleware (AOM)

Bau auf KOM auf.
Ist komplexer und bietet mehr Dienste.
Übernimmt Vewaltung von Ressourcen als Laufzeitumgebung

- Threading
- Getrennte Speicherbereiche
- Verbesserung der Performance
- Hohe Verfügbarkeit
- Zugriffskontrolle und Zugriffsrechte

Unterschiedliche Middlware-Impl stellen unterschiedliche Dienste zur verfügung.

- Namensdienst
  - Veröffentlichung Ressourcen z.B. im Internet
  - DNS
- Sitzungsverwaltung
  - Session Hanlding
- Transatkionsverwaltung
  - Parallele Zugriffe
- Persistenz
  - OR Mapper

Typen von Technologien

- Object Request Broker (ORB)
  - Nutzt RPC
- Application Server (AS)
  - Tomcat, Jetty
- Middlware-Plattformen
  - JEE mit dem EJB-Kompontenmodell
  - .NET mit COM-Kompontentenmodell

Remote Methode Invocation

02 April 2015 13:43

Verwaltet entferne Objekte
Einfaches Framework für verteilte Java Anwendungen
Java to Java only
Client

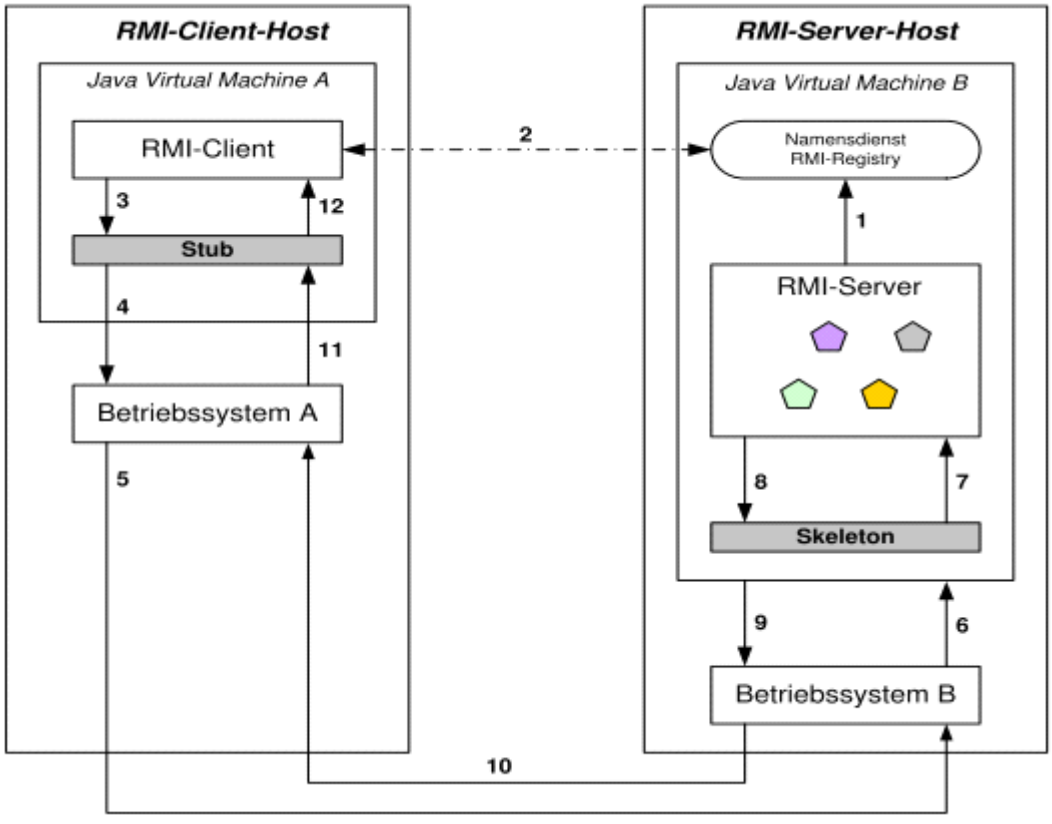
- Nimm Dienst von Objekt in Anspruch
- Zugriff über Namensdienst
- Aufruf von Methoden des RO

Server

- Stellt Objekt bereit
- Registration beim Namensdienst

Remote Object

Vom Server ezeugt
Registriert beim Namensdienst
Läuft auf Server
Stellt Dienste zur Verfügung
Client-Aufrufe empfindet als lokale Aufrufe



Stub und Skeleton werden von RMI generiert.
Sub-Objekt

- Nimmt Client-Anfragen entgegen
- Verpackt Parameter (marshalling) und sendet Methodenaufrufe

Skeleton-Objekt

- Verarbeitet Anfrage und macht Methodenaufruf

Implementierungx

Zur Vereinheitlichung werden Schnittstellen benutzt.

Beispiel:

```
import java.rmi.*;

public interface Adder extends Remote {
       int add(int x, int y) throws RemoteException;
}
```

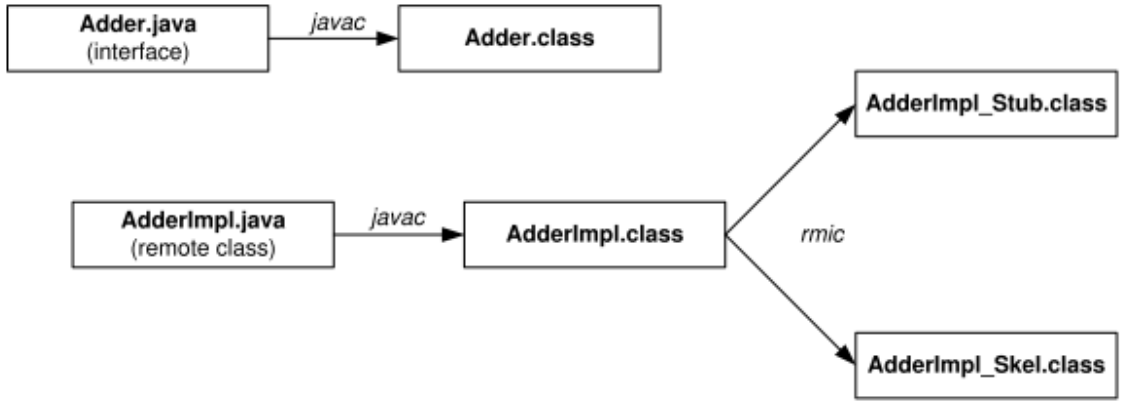
Der Stub und das Skeleton werden mit rmic-Tool (RMI Compiler) erstellt.

Das Objekt besteht aus Schnittstelle und Impl.

Beispiel:

```
import java.rmi.*;
import java.rmi.server.*;

public class AdderImpl extends UnicastRemoteObject
implements Adder {

    public AdderImpl() throws RemoteException {

    }

    public int add(int x, int y) throws RemoteException {
        return x + y;
    }
}
```

Dazu ein Überblick:



Das entfernte Objekt muss dann noch beim Namensdienst registriert werden.
Namendienst starten:

```
start rmiregistry <port>
```

Das ist das grundlegende Konzept, zur Verwendung braucht es aber noch ein Server und ein Client.

Beispiel - Server:

```
public static void main(String[] args){

    try {
        // Entferntes Objekt erzeugen
        Adder adder = new AdderImpl();

        // Entferntes Objekt beim Namensdienst registrieren
        Naming.rebind("AdderObjekt", adder);

        // Ausgabe - Server bereit
        System.out.println("Adder bound");
    } catch (RemoteException re){
        re.printStackTrace();
    } catch (MalformedURLException me){
        me.printStackTrace();
    }

}
```

RMI-Client

Muss SecurityManager installieren -> schutz vor malicious Stubs.
Namensdienst und Objekt lookup

`rmi://192.168.1.25:1099/AdderObjekt`

Entfernte Schnittstelle casten

```
public class Main {

    public static void main(String[] args) {

        // SecurityManager installieren
        System.setSecurityManager(new RMISecurityManager());

        // URL definieren
        String url = "rmi://196.168.1.25:1099/AdderObjekt";

        // Referenz auf das entfernte Objekt holen
        Adder adderObj = (Adder) Naming.lookup(url);

        // Methode 'add' des entfernten Objekts aufrufen
        int sum = adderObj .add(13, 37);

        // Ergebnis ausgeben
        System.out.println("13 + 37 = " + sum);
    }
}
```

Client starten:

`java -Djava.security.policy=adder.policy Client`

Security

Über Policy Datei werden Anweisungen festgehalten und auf dem Client hinterlegt.
Diese Datei legt fest welche Ausführungen erlaubt sind.

Beispiel - Policy:

```
grant {
  permission java.net.SocketPermission „*:1024-", "connect, accept";
};
```

Web Services

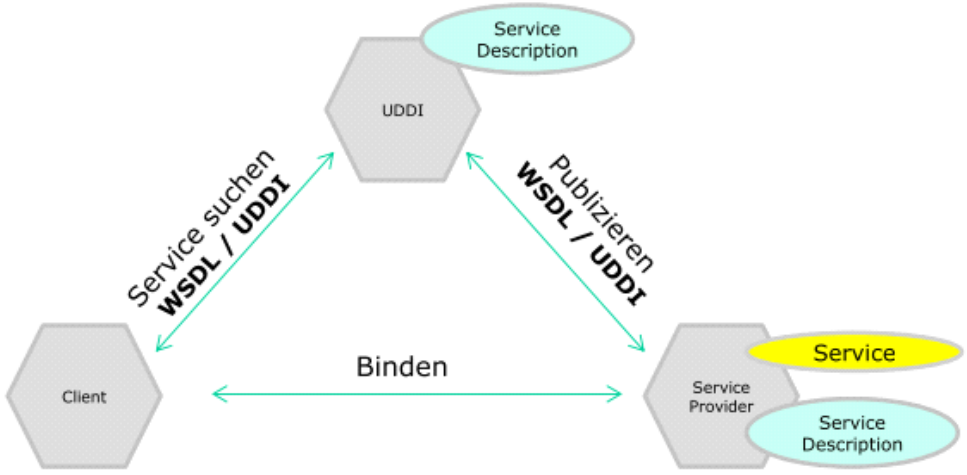
09 April 2015 13:22

- Dienst der über ein Netzerk per URL erreichbar ist.
- Besteht aus Komponenten, die verschiedene Funktionen zur Verfügung stellen.
- Unterstützen die Zusammenarbeiter von verschieden Anwendung auf unterschieldiche plattformen und/oder Frameworks.

Grundelemente (Plattform)

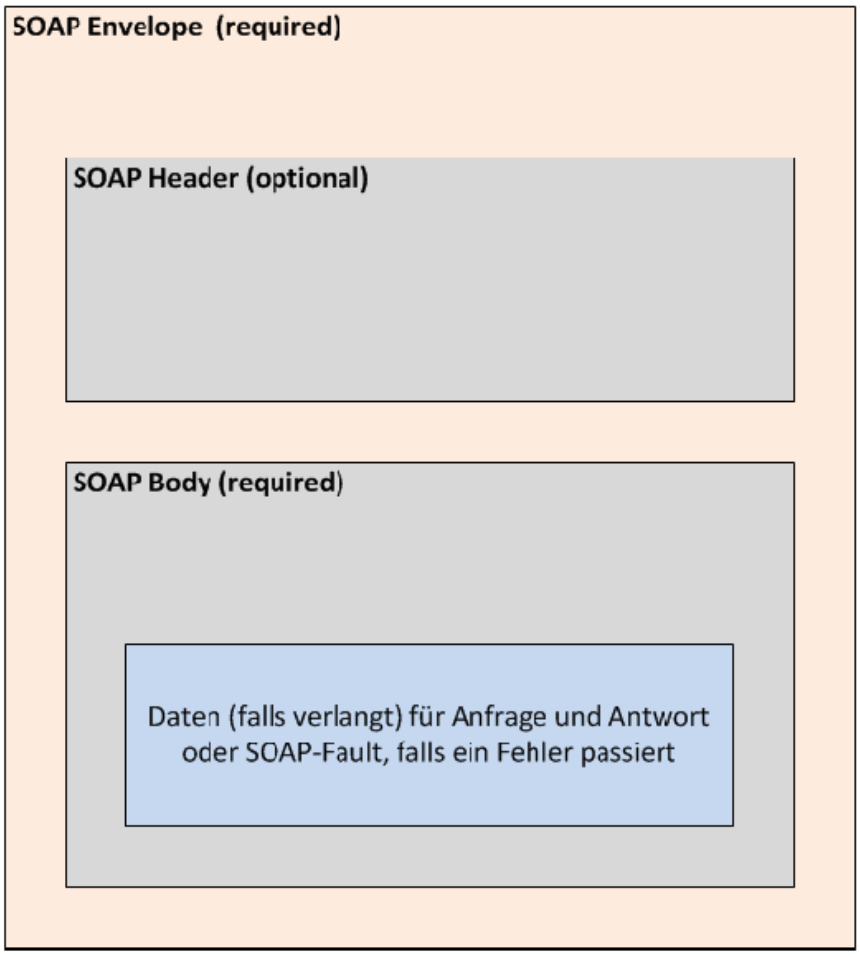
- SOAP (**S**imple **O**bject **A**ccess **P**rotocol)
- UDDI (**U**niversal **D**escription, **D**iscovery und **I**ntegration)
- WSDL (**W**eb **S**ervice **D**escription **L**anguage)

Web Services Architektur:



SOAP

- Leichtgewichtiges Kommunikationsprotokoll
- Basiert auf XML
- Plattform- und Programmiersprachunabhängig.
- Definiert die Struktur der Nachricht
  - Regeln
  - Sytanx



- Envelope
  - Markiert das XML Dokument als SOAP Nachricht
  - Definiert Namespace
- Header
  - Optional
  - Anwendungsspezfische Daten (Auth, Resource, ..)
- SOAP-Body
  - Enthält Nutzerdaten

Beispiel - SOAP Request:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

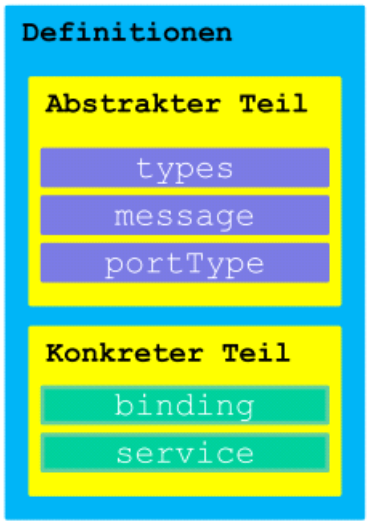
Beispiel - SOAP Response:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

WSDL

- Beschreibt einen Web Service
- Basiert auf XML
- Ist eine Metasprache und beschreibt
  - Funktionen
  - Daten
  - Datentypen
  - Austauschprotokolle
- Wird benutzt um Web service zu finden
- WSDL Dokument besteht aus
  - types
  - messages
  - portType
  - binding

- service



Diese WSDL-Teile werden verknüpft.

Java Web Services

09 April 2015 13:43

Für den WSDL Service wird Metro (RI für JAX-WS) benutzt.
RI ist eine Referenz Implementierung.

Beispiel WSDL

Interface:

```
package time.model;

import javax.jws.*;
import java.util.List;

@WebService
public interface Time {
      @WebMethod
      long getCurrentTime(@WebParam(name = "cityName") String cityName);

      @WebMethod
      List<String> getAvailableCityNames() throws Exception;
}
```

Klasse:

```
package time.business;

import javax.jws.WebService;
import time.model.Time;

@WebService(endpointInterface = "time.model.Time")
public class TimeImpl implements Time {
    public long getCurrentTime(String cityName) {
        // Implementierung ...
    }

    public List<String> getAvailableCityNames() throws Exception {
        // Implementierung ...
    }
}
```

Webserice publizieren:

```
package time;

public class Publisher {
    public static void main(String[] args) {
        // Service-Objekt erstellen
        Time service = new TimeImpl();
        // URI definieren
        String uri= "http://localhost:9090/time";
        // WebService publizieren
        Endpoint ePoint = Endpoint.publish(uri,service);
        // Dialog zum Beenden des WebServices anzeigen
        JOptionPane.showMessageDialog(null, "Server beenden");
        // WebService-Ausführung beenden
        ePoint.stop();
    }
}
```

Unter http://localhost:9090/time?wsdl findet man ein WSDL Dokument, das den Service beschreibt.

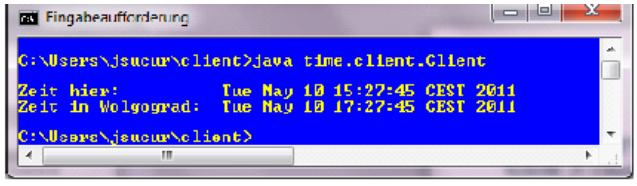
Auf dem Client kann das Dokument importiert werden.

```
wsimport -keep http://localhost:9090/timeService?wsdl
```

Dabei generiert man eine Stub-Klasse bzw. handelt es sich um ein Client-Artefakt.

Client-Klasse:

```
package time.client;

import java.util.*;
import time.business.Time;
import time.business.TimeImplService;

public class Client {
    public static void main(String[] args) {
        /* Service kreieren */
        TimeImplService service = new TimeImplService();

        /* Proxy kreieren (Client-Stub) */
        Time proxy = service.getTimeImplPort();

        /* Aktuelle Zeit in Wolgograd abfragen */
        long timeInMillis = proxy.getCurrentTime("Wolgograd");
        Date d = new Date(timeInMillis);

        /* Ausgabe */
        System.out.println("Zeit hier:         " + new Date());
        System.out.println("Zeit in Wolgograd: " + d);
    }
}
```



SOAP

Unterstützt Asynchrone Aufrufe über das Callback Prinzip
Kann mit diversen Server betrieben werden (Tomcat, Jboss)

Tomcat

Ist eine RI des Servlet-Container.

Führt WAR Dateien aus.

Beispiel - Web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
        <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>
      <listener>
            <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
      </listener>
      <servlet>
            <servlet-name>TimeServlet</servlet-name>
            <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>TimeServlet</servlet-name>
            <url-pattern>/*</url-pattern>
      </servlet-mapping>
      <session-config>
            <session-timeout>180</session-timeout>
      </session-config>
</web-app>
```

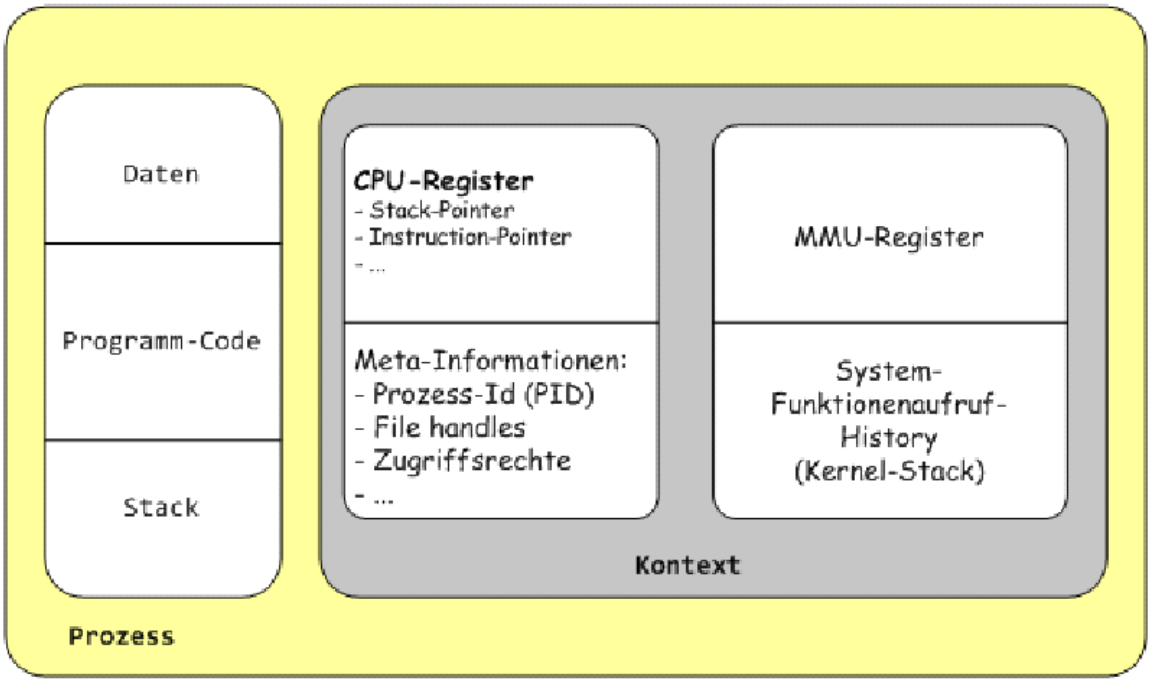
Multi-Threading

16 April 2015 13:23

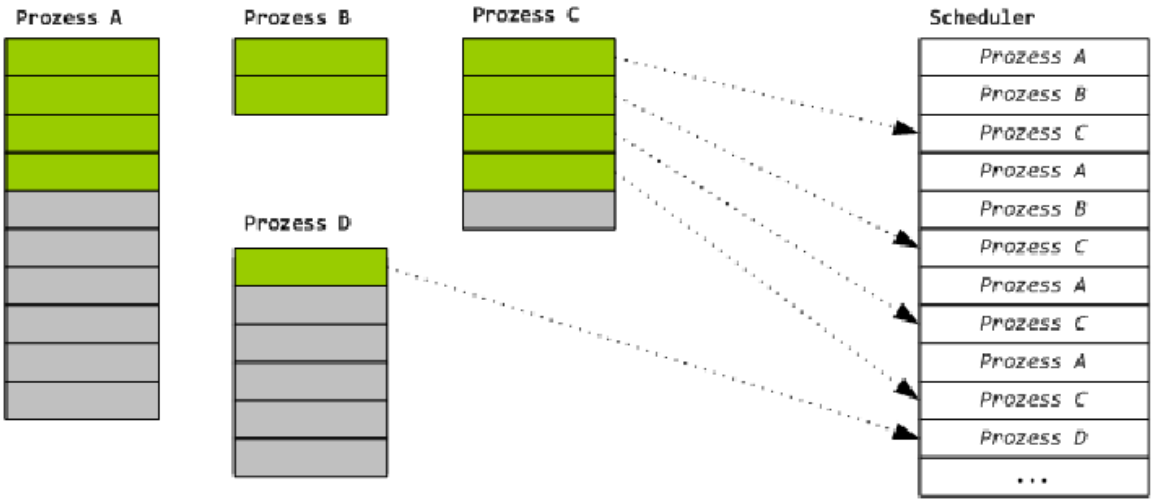
- Ein Prozess hat mehrere Kontrollflüsse, die parallel ausgeführt wird.
- Vorraussetzung
  - Mehrkernprozessoren
  - Unterstützung Betriebssystem
- Time-Sharing Betriebssysteme erlaubte erstmals eine parallele Ausführung

Prozess

- Ist ein Programm in Ausführung.
- Prozess muss unterbrechbar sein.
  - Für kurze Unterbrechungszeit werden Resourcen anderen Prozessen zur Verfüung gestellt.
  - Prozess darf von Unterbrechung nichts mitbekommen
- Ein Prozess hat eine Umgbung mit entsprechenden Daten
  - Prozessspezifische Daten (Id, Priorität, …)
  - Code zum Ausführen
  - Ablage für die während Laufzeit generierten Daten (Stack)
  - Registerinhalte: Stack-Pointer (SP), Instruction-Pointer (IP) usw.
  - Diverse Meta-Informationen
  - Die für MMU relevanten Informationen

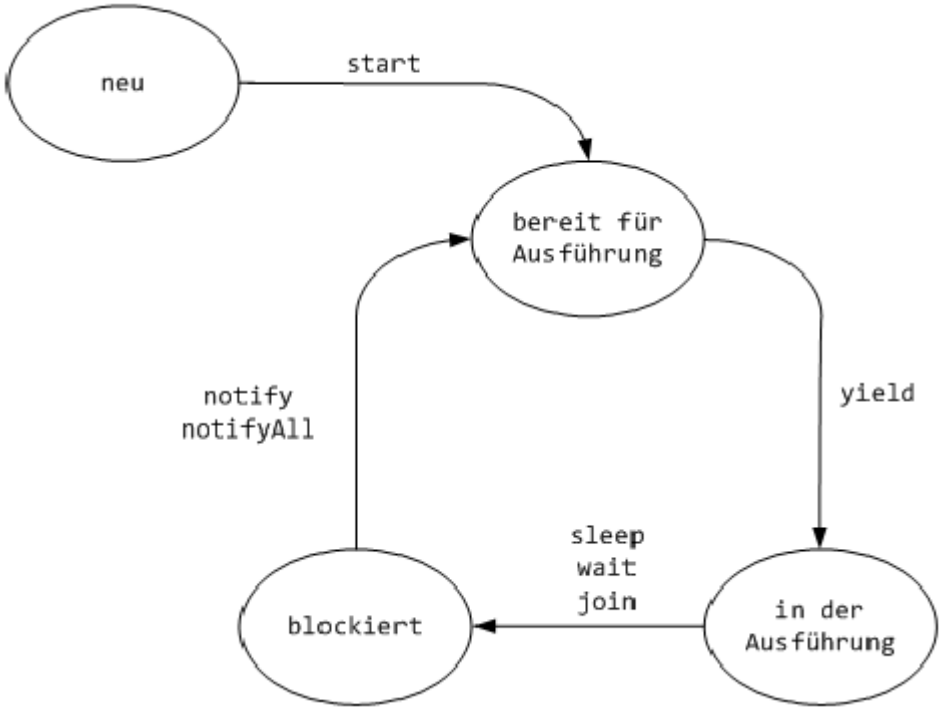


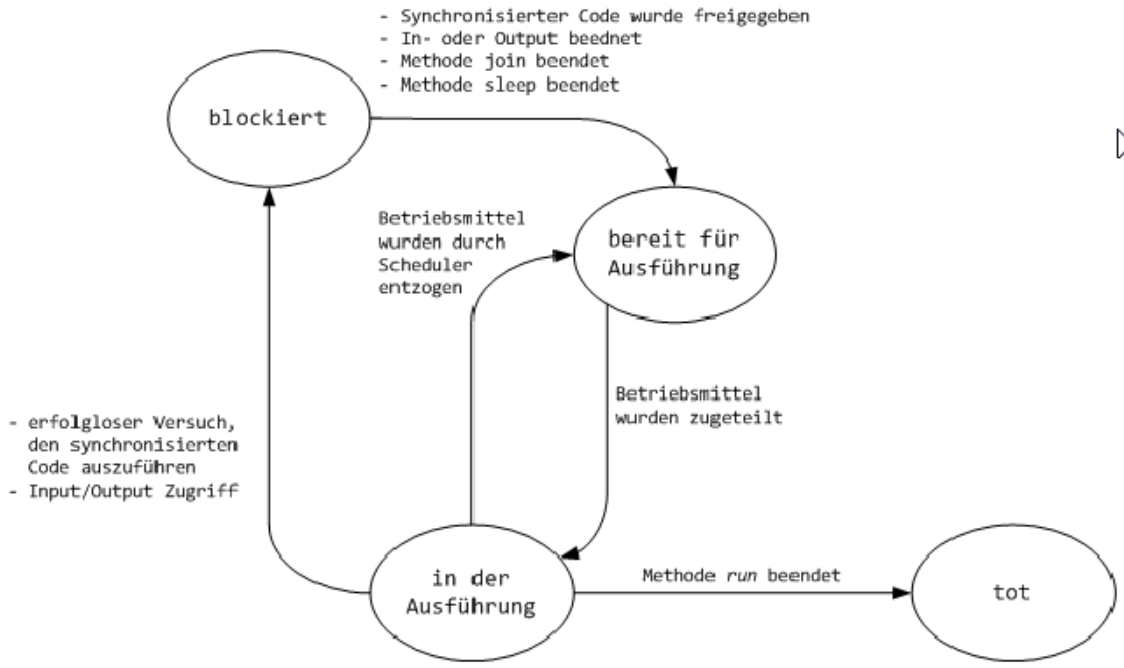
Bei jedem Wechsel durch den Scheduler wird der gesamte Prozess-Kontext ausgelagert.



Threading

- Ist ein leichtgewichtiger Prozess
- Der Kontextwechsel ist nicht aufwendig
- Ein Prozess kan mehrere Threads starten
- Ein Thread hat eigenen
  - Stack
  - Befehlszeiger
  - Stakzeiger
  - Satz von Rgister
- Gemeinsamt haben Threads
  - Programmcode
  - Prozessspezifische Daten
  - Datei-Informationen
  - globale Daten
- Ein Thread ist entweder
  - neu
  - bereit Ausführung
  - in Ausführung
  - blockiert
  - tot



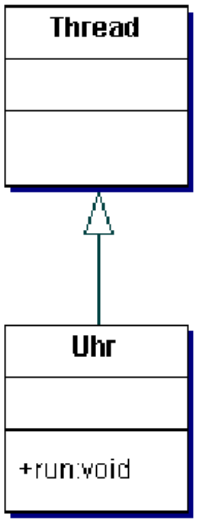

- Synchronisierter Code wurde freigegeben
- In- oder Output beednet
- Methode join beendet
- Methode sleep beendet
blockiert
Betriebsmittel wurden durch Scheduler entzogen
bereit für Ausführung
Betriebsmittel wurden zugeteilt
- erfolgloser Versuch, den synchronisierten Code auszuführen
- Input/Output Zugriff
in der Ausführung
Methode run beendet
tot

Threads in Java

16 April 2015 13:38

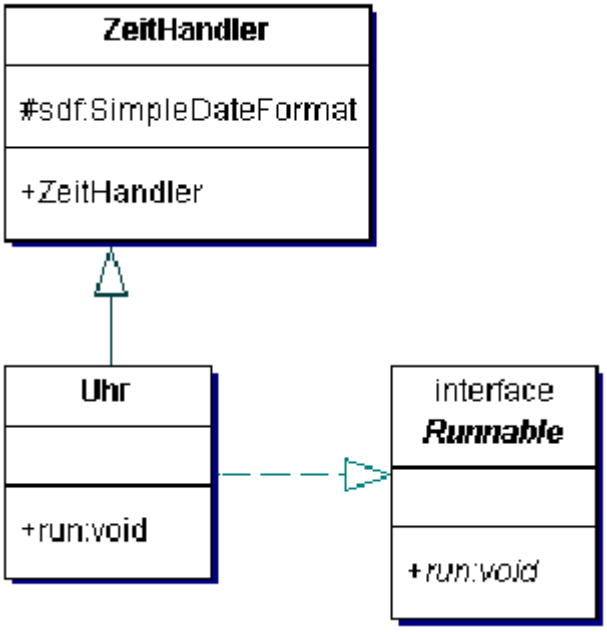
- Klasse java.lang.Thread
- Schnittstelle java.lang.Runnable
- Über beide Resourcen kann ein Objekt threadfähig gemacht werden.

Beispiel - Thread:




```
import java.text.SimpleDateFormat;
import java.util.Date;

public class Uhr extends Thread {
     public void run() {
          SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss:SSS");

          while (true) {

               try {
                    Date d = new Date();
                    System.out.println("Zeit: " + sdf.format(d));
                    Thread.sleep(1000);
               } catch (InterruptedException e) {
                    // Ausnahmebehandlung ...
               }
          }
     }
}

  public class Main {

       public static void main(String[] args) {
            // Klasse Uhr instanzieren
            Uhr timeThread = new Uhr();

            // Thread-Ausfuehrung starten
            timeThread.start();
       }
  }
```

Beispiel - Runnable:




```java
import java.text.SimpleDateFormat;

public class ZeitHandler {

     protected SimpleDateFormat sdf = null;

     public ZeitHandler() {

          // SimpleDateFormat erstellen
          sdf = new SimpleDateFormat("dd.MM.yyyy 'at' hh:mm:ss");
     }
}

public class Uhr extends ZeitHandler implements Runnable {

      public void run() {

           while (true) {
                try {
                     String dateAsString = sdf.format(new java.util.Date());
                     System.out.println("Date & Time: " + dateAsString);
                     Thread.sleep(1000);
                } catch (InterruptedException e) {
                     // Ausnahmebehandlung …
                }
           }
      }
}
```

```
public class Main {

     public static void main(String[] argv) {

          // Ein Runnable-Objekt erzeugen
          Uhr runnableObj = new Uhr();

          // Einen Thread erzeugen, wobei dem Konstruktor als
          // Parameter ein Runnable-Objekt uebergeben wird
          Thread timeObj = new Thread(runnableObj);

          // Ausfuehrung starten
          timeObj.start();
     }
}
```

Interrupting

Über Methode interrupt und istInterrupted kann der Zustand abgefragt werden.
Der Aufruf löst eine Exception aus.

Beispiel - interrupt:

```
public class Main {
     public static void main(String[] args) {
          // Klasse Uhr instanzieren
          Uhr timeThread = new Uhr();

          // Thread-Ausfuehrung starten
          timeThread.start();

          try {
               // Haupt-Thread 30 Sekunden schlaffen lassen
               Thread.sleep(30000);

               // Den timeThread stoppen
               timeThread.interrupt();

          } catch (InterruptedException e) {
               // Ausnahmebehandlung …
          }
     }
}
```

Beispiel - Interrupted:

```
import java.text.SimpleDateFormat;

public class Uhr extends Thread {

     public void run() {

          SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss:SSS");

          try {

               while (!isInterrupted()) {
                    String dateAsString = sdf.format(new java.util.Date());
                    System.out.println("Zeit: " + dateAsString);
                    Thread.sleep(1000);
               }

          } catch (InterruptedException e) {
               // Ausnahme behandeln falls nötig bzw. sinnvoll ...
          }
     }
}
```

Method interrupt setzt Interrupted-Flag auf true
isInterrupted liefert Wert Flag zurück.

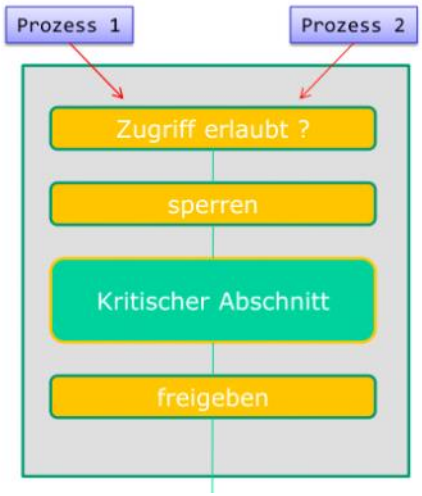
Der Thread kann nun wie folgt beendet werden:

```
public class Uhr extends Thread {

      private volatile boolean beendet = false;

      public void run() {
            java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat("hh:mm:ss:SSS");

            while (!beendet) {
                  try {
                        String dateAsString = sdf.format(new java.util.Date());
                        System.out.println("Zeit: " + dateAsString);
                        Thread.sleep(1000);
                  } catch (InterruptedException e) {
                        // Ausnahmebehandlung …
                  }
            }
      }

      // Methode beenden
      public void beenden() {
            beendet = true;
      }
}
```

Gemeinsame Ressourcen

- Der Zugriff muss geregelt werden, damit Anomalien vermieden werden.
- Beim Zugriff herrscht race condition
- in kritischen Abschnitten (shared Access) darf nur ein Prozess laufen
- Prozess muss vor Ausführung um erlaubnis fragen.

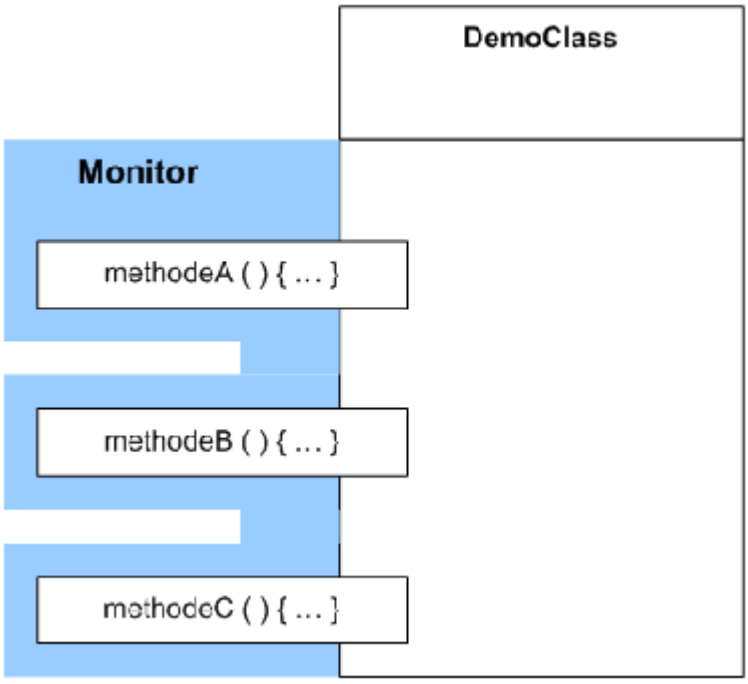
- Eingang-Sektion:
  - Eintrittserlaubnis
  - Eintritt sperren
- Kritischer Abschnitt
  - Zugriff auf gemeinsame Ressourcen
- Ausgang-Sektion
  - Eintritt freigeben



- Über das Monitorkonzept können Zugriffe synchronisiert werden
- Schlüsselwort synchronized

```
public class DemoClass{
    public static synchronized void methodeA() {
        // kritischer Abschnitt
    }

    public static synchronized void methodeB() {
        // kritischer Abschnitt
    }

    public static synchronized void methodeC() {
        // kritischer Abschnitt
    }

    // Weitere konkrete Methoden …
}
```

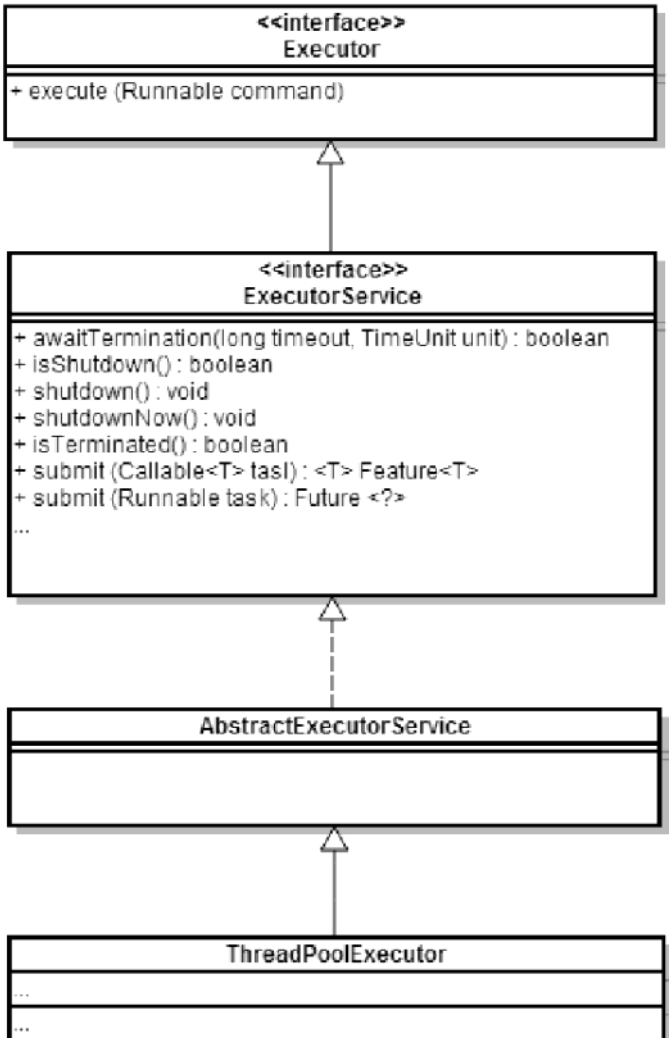
- Der Monitor überwacht die Zugriffe



- Zur Synchronisierung wird ein Schlüssel benötigt.
- Der Schnüssel steuert den Zugriff auf mehrere Codeblöcke (Schlüsselbrett = Monitor)
- Anzahl Monitore -> Anzahl Schlüssel für Synchronisation.
- Der Schlüssel kann sein
    - Instanz der Klasse Class
    - Referenz auf das aktuelle Objekt
    - übergebenes Schlüssel-Objekt

Executor Framework und Thread Pool

- Thread_pool enthält laufene Threads, die Arbeiten asynchron erledigen.
- Vorteile
    - Wenig Overhead bei Mutation Thread
    - Aufgaben werden eingereiht und nicht direkt auf Thread abgebildet



Beispiel - Executor Framework und Thread-Pools:

Die auszuführende Klasse:

```
public class PrimeFactorsPrinter implements Runnable {

    private long number;

    public PrimeFactorsPrinter(long number) {
        this.number = number;
    }

    @Override
    public void run() {
        long currentValue = number;
        List<Long> prims = new ArrayList<>();
        for (long i = 2; i <= currentValue; i++) {
            if (currentValue % i == 0) {
                prims.add(i);
                currentValue /= i;
                i = 1;
            }
        }

        show(prims); // Resultat anzeigen (Code hier nicht angegeben)
    }
}
```

Die Main Methode mit Executor:

```java
import java.util.concurrent.*;

public class Main {

    public static void main(String[] args) throws InterruptedException {
        /* Anzahl Threads */
        final int NUMBER_THREADS = 10;

        /* ExecutorService erstellen */
        ExecutorService executor =
            Executors.newFixedThreadPool(NUMBER_THREADS);

        for (long i = 1000000001; i < 1000000050; i++) {
            executor.execute(new PrimeFactorsPrinter(i));
        }

        executor.shutdown();
    }
}
```

- Probleme Runnable hat void als Rückgabewert.
- Mit Callable-Objekt wird dies Umgangen
- Es liefert ein Fture Objekt zurück und ermöglich die Abfrage des Task-Zustandes

Beispiel - Callable und Future:

Klasse zum Sammeln von Primfaktoren

```java
public class PrimeFactorsFinder<V> implements Callable<List<Long>> {

    private Long n;

    public PrimeFactorsFinder(Long n) { this.n = n; }

    @Override
    public List<Long> call() throws Exception {

    List<Long> primeFactorsList = findPrimeFactors(n);
        return primeFactorsList;
    }

    private List<Long> findPrimeFactors(long n) {
        List<Long> prims = new ArrayList<>();
        // Implementierung ...
        return prims;
    }
}
```

Main Methode mit thread pools.

```
public class Main {

    public static void main(String[] args) {

        long startValue = 100000001;
        long endValue = 100000050;

        ExecutorService executor = Executors.newFixedThreadPool(5);

        for (Long i = startValue; i < endValue; i++) {
            Callable<List<Long>> c = new PrimeFactorsFinder<Long>(i);
            Future<List<Long>> prims = executor.submit(c);
            show(i, prims);
        }

        executor.shutdown();
    }

    // Implementierung der Methode show …
}
```

Show Methode gibt laufend primfaktoren aus.

```
 private static void show(long number, Future<List<Long>> future) {
     try {
         List<Long> prims = future.get();

         if (prims.size() == 1) {
             System.out.println(number + ": PRIMZAHL");
         } else {
             String str = number + " = " + prims.get(0).longValue();

             for (int i = 1; i < prims.size(); i++) {
                 str += " * " + prims.get(i).longValue();
             }

             System.out.println(str);
         }

     } catch (InterruptedException | ExecutionException e) {
         // Ausnahme behandeln …
     }
 }
```

- Ist gute Lösung in Applikation mit kurzlebigen und unabhängigen Aufgaben
    - Z.B. Server mit Pool für Client-Anfragen