OOAD - Zusammenfassung

Jürg & DIE Schweisser, Hannes, Gwerder, Körner, Niedermann

9. September 2013

Inhaltsverzeichnis

1 C++ CheatSheet **2**
1.1 Header Files . . . 2
1.2 Unterschied Statisch - Dynamisch . . . 2
1.3 Assoziationen . . . 2
1.3.1 „0..1:0..1“ . . . 2
1.3.2 „1:0..1“ . . . 2
1.3.3 „0..1:0..n“ . . . 3
1.4 Vererbung . . . 3

2 OOA **4**
2.1 Ziel der Analyse . . . 4
2.2 Ziel der OOA . . . 4
2.3 OOA Modelle . . . 4
2.4 Attribut Balzert S. 26 . . . 4
2.5 Klasse Balzert S. 23 . . . 5
2.5.1 Notation . . . 5
2.6 Objekt Balzert S. 20 . . . 5
2.6.1 Notation . . . 5
2.6.2 Geheimnisprinzip . . . 5
2.6.3 Objektidentität . . . 5
2.7 Assoziation Balzert S. 42 . . . 5
2.8 Generalisierung (Vererbung) Balzert S. 52 . . . 6
2.9 Aktivität Balzert S. 69 . . . 6
2.10 Zustandsautomat Balzert S. 87 . . . 6
2.11 Objektdiagramm (*object diagram*) Balzert S. 21 . . . 7
2.12 Use-Case Diagramm Balzert S. 62 . . . 7
2.13 Szenario Balzert S. 80 . . . 7
2.14 Analyseprozess Balzert S. 130 . . . 7

3 OOD **8**
3.1 Ziel des Entwurfs . . . 8
3.2 Klassen Balzert S. 252 . . . 8
3.3 Attribut Balzert S. 261 . . . 8
3.4 Operationen Balzert S. 266 . . . 9
3.5 Komponente Balzert S. 273 . . . 9
3.6 Assoziation Balzert S. 284 . . . 9
3.7 Polymorphismus Balzert S. 293 . . . 9
3.8 Generalisierung Balzert S. 300 . . . 10
3.9 Szenario Balzert S. 327 . . . 10
3.10 Entwurfsmuster Balzert S. 355 . . . 10

1 C++ CheatSheet

1.1 Header Files

Include-Guard:

```
#ifndef MeinHeader_H
#define MeinHeader_H
// Ganzer Inhalt vom Header
#endif
```

Header und using namespace:
Kein using namespace std; Wenn eine Funktion aus dem diesem Namespace gebraucht wird kann mit dem vollen Namen darauf zugegriffen werden. (std::cout << "..." << std::endl;)

1.2 Unterschied Statisch - Dynamisch

Statisch:

```
ClassA a;
a.foo(); // Funktionsaufruf
bar(&a); // Uebergabe von this
```

Dynamisch:

```
ClassA* a = new A();
a->foo(); // Funktionsaufruf
bar(a); // Uebergabe von this
```

1.3 Assoziationen

1.3.1 „0..1:0..1“

Vorwärtsdekleration wird gebraucht, wenn zwei Klassen sich gegenseitig referenzieren.

Header A:

```
class B; // Vorwaertsdeklaration

class A{
    public:
        A();
        void setB(B* pB);
        B* getB();

    private:
        B* mB; // Pointer auf B-Objekt
};
```

Header B:

```
class A; // Vorwaertsdeklaration

class B {
    public:
        B();
        void setA(A* pA);
        A* getA();

    private:
        A* mA; // Pointer auf A-Objekt
};
```

Implementation:

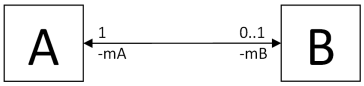
```
void A::setB(B* pB) {
    mB = pB;
}
```

Aufruf:

```
A a;
B b;
a.setB(&b);
```

1.3.2 „1:0..1“

Eine 1:0..1 Assoziation kann ähnlich wie eine 0..1:0..1 Assoziation implementiert weden. Wobei auf der „1 Seite“ die andere Klasse direkt im Konstruktor mitgegeben wird. Ein Setter wird auf der B-Seite nicht mehr gebraucht! Achtung: untenstehende Implementation zeigt nur schematisch auf wie es funktioniert. Copy- und Zuweisungskonstruktor müssten auch noch korrekt implementiert werden.



Header B (Muss Assoziation)

```
// ...
class B {
    public:
        A* getA();
        B(A* pa);          // Konstruktor

    private:
        A* mA;
        B();               // Default Konstruktor
}
```

Implementation B (Muss Assoziation)

```
B::B(A* pA)
: mA(pA)
{
    mA->setB(this);
    // Kann auch im im Aufruf erfolgen
}
```

Aufruf:

```
    A a;
    B b(&a);
```

1.3.3 „0..1:0..n“

Header A:

```
class B; // Vorwaertsdeklaration

class A{
    private:
        B* mB[n]; // Pointerarray auf B-Objekte

    public:
        A();
        void addB(B* pB);
        void removeB(B* pB);
};
```

Header B:

```
class A; // Vorwaertsdeklaration

class B {
    private:
        A* mA; // Pointer auf A-Objekt

    public:
        B();
        void setA(A* pA);
        A* getA();
};
```

Implementation:

```
A::A() {
    for(int i = 0; i<n; i++) {
        // Alle Werte auf 0 initialisieren
        mB[i] = 0;
    }
}

// fuer removeB sinngemaess (->setA(0); mB[i]=0)
void A::addB(B* pB) {
    for(int i = 0; i<n; i++) {
        if(mB[i] == 0) {
            mB[i] = pB;
            mB[i]->setA(this);
            return;
        }
    }
}
```

Aufruf:

```
A a;
B b1;
B b2;
a.addB(&b1);
a.addB(&b2);
a.removeB(&b2);
// ...
```

1.4 Vererbung

Basisklasse

```
    class: Fahrzeug
    {
    public:
             Fahrzeug(int pGewicht);
        void setGewicht(int pGewicht);

    private:
        int mGewicht;
    };
```

abgeleitete Klasse

```
    class: Auto : public Fahrzeug
    {
    public:
             Auto(int pPersonen, int pGewicht);
        void setPersonen(int pPersonen);

    private:
        int mPersonen;
    }
```

Aufruf

```
    Auto.setGewicht(42);
    Auto.setPersonen(4);
```

2 OOA

2.1 Ziel der Analyse

- Wünsche und Anforderungen an ein neues System ermitteln und beschreiben
- Bei Modellbildung bewusst **alle** Aspekte der Implementierung ausklammern
- System als ideal anschauen. D.h keine Verzögerungen, keine Fehler, unendlicher Speicher

2.2 Ziel der OOA

Ziel ist es, das zu realisierende Problem zu verstehen und in einem OOA-Modell zu beschreiben.

2.3 OOA Modelle

statisches Modell

beschreibt ...

... die Klassen des Systems

... die Assoziation zwischen den Klassen

... die Generalisierung (Vererbung)

ausserdem enthält es die Daten des Systems (Attribute)

dynamisches Modell

- zeigt die Funktionsabläufe
- Use-Cases beschreiben die durchzuführenden Aufgaben
- Aktivität beschreibt die Ausführung von Funktionalität und Verhalten
- Szenario zeigen wie Objekte miteinander kommunizieren
- Zustandsautomaten beschreiben in der Analyse die Reaktion eines Objektes auf verschiedne Ereignisse

OOA-Modell

- beschreibt die essenzielle Struktur und Semantik des Problems, aber noch keine technische Beschreibung
- enthält keinerlei Optimierung für das verwendete Computersystem oder die benutzte Basissoftware
- bildet fachliche Lösung des zu realisierenden Systems

primitive Datentypen

- Boolean
- String
- Integer
- UnlimitedNatural

2.4 Attribut Balzert S. 26

Attribut Beschreibt Daten die von den Objekten der Klasse angenommen werden können. Alle Objekte einer Klasse haben dieselben Attribute, können aber unterschiedliche Attributwerte haben.

Klassenattribut Wenn nur ein Attributwert für alle Objekte einer Klasse existiert. Sie existieren auch, wenn es zu einer Klasse noch keine Objekte gibt. Wird unterstrichen

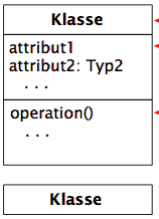
abgeleitets Attribut kann zu jeder Zeit aus anderen Attributwerten berechnet werden. Wird mit **/zahl** geschrieben.

Eigenschaftswerte siehe Balzert S.29

Multiplizität [0..n] → spezifiziert, [0..*] → unspezifiziert

2.5 Klasse Balzert S. 23

Eine Klasse ist eine Vorlage für ein Objekt.



Operation Eine Funktion die auf alle Attributwere eines Objekts Zugriff hat.

Klassenoperation Eine Operation, die der jeweiligen Klasse zugeordnet ist. Kann nicht auf ein einzelnes Objekt der Klasse angewendet werden. Wird unterstrichen

Verhalten Das Verhalten der Klasse ist die Menge aller Operationen

Abstrakte Von einer abstrakten Klasse können keine Objekte erzeugt werden. (*kursiver Klassenname* oder {abstract})

Basisklasse Vererbt abgeleiteten Klassen Attribute und Operationen

2.5.1 Notation

Der **Klassenname** ist immer ein Substantiv im Singular, das durch ein Adjektiv ergänzt werden kann. Der Klassenname wird fett geschrieben und beginnt mit einem Grossbuchstaben.

2.6 Objekt Balzert S. 20

Ein Objekt wird aus einer Klasse erzeugt, ist also ein Exemplar einer Klasse.

Zustand Bestimmt durch seine Attributwerte und seine Objektbeziehungen zu anderen Objekten

Verhalten Die beobachtbaren Effekte aller Operationen bestimmt durch die Operationsaufruf, auf die diese Klasse bzw. deren Objekte reagieren.

Objektidentität jedes Objekt hat eine Identität, welche einzigartig ist (unique)

2.6.1 Notation

Objektnamen werden immer klein geschrieben und unterstrichen.

:Klasse	anonymes Objekt
objekt:Klasse	Objekt wird über Namen angesprochen
objekt	wenn Objektname zur Identifikation ausreicht

2.6.2 Geheimnisprinzip

Daten sind nach aussen verborgen und können nur über Operationen gelesen oder geändert werden.

2.6.3 Objektidentität

Eigenschaft, die ein Objekt von allen anderen Objekten unterscheidet. Bei zwei Objekten mit gleichen Attributwerten sprechen wir von Gleichheit, wenn dasselbe Objekt gemeint ist von Identität.

2.7 Assoziation Balzert S. 42

Modelliert Objektbeziehungen zwischen Objekten einer oder mehrerer Klassen. Jede Assoziation wird durch Mutliplizitäten, einen optionalen Namen oder Rollennamen beschrieben.

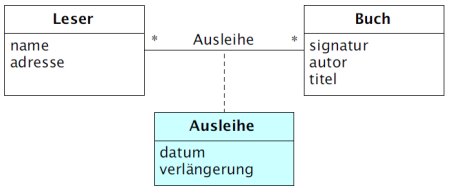
binäre Assoziation Assoziation zwischen 2 Objekten

ternäre Assoziation Assoziation zwischen 3 Objekten

n-äre Assoziation zwischen n Objekten

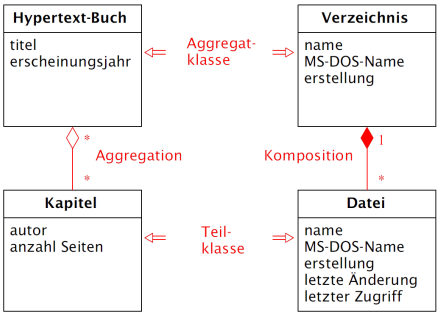
reflexive Assoziation Verbindet zwei Objekte der gleichen Klasse

Assoziationsklasse Besitzt die Eigenschaften einer Assoziation und einer Klasse

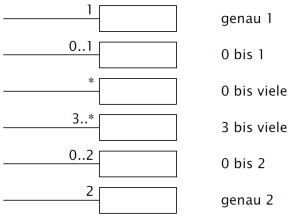


Aggregation Sonderfall der Assoziation. „ist Teilvon“ oder „besteht aus“.

Komposition starke Form der Aggregation. Beim löschen werden alle Teile gelöscht werden. Jedes Teil kann nur zu einem Ganzen gehören



Multiplizität Bezeichnet die Anzahl der an der Assoziation beteiligten Objekte.



abgeleitete Assoziation Die Abhängigkeiten sind bereits durch andere Assoziationen beschrieben worden. Präfix „/“

Assoziationsname Beschreibt im Allgemeinen nur eine Richtung der Assoziation.

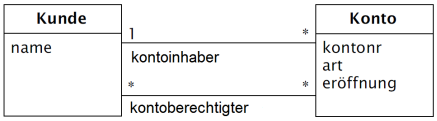
Leserichtung Angabe beim Assoziationsnamen. Wird mit ausgefülltem schwarzem Dreieck (▶) dargestellt.

Sichtbarkeit Vor dem Rollenname geschrieben. -private #protected +public ∼package

Eigenschaftswert OOA: siehe Balzert S.45

Rolle Information oder die Bedeutung/Funktion einer Klasse in Beziehung zu einer anderen.

Rollenname Jeweils am Ende der Assoziationsrichtung von der Klasse, deren Bedeutung sie beschreibt. Kann zur Verständlichkeit beitragen.



2.8 Generalisierung (Vererbung) Balzert S. 52

Beschreibt die Beziehung zwischen einer allgemeinen Klasse (Basisklasse) und einer spezialisierten Klasse. Die spezialisierte Klasse ist vollständig konstistent mit der Basisklasse, enthält aber zusätzliche Informationen (Attribute, Operationen, Assoziationen). Jedes Objekt der Unterklasse **ist ein** Objekt der Oberklasse.

Einfachvererbung erbt von einer Basisklasse

Mehrfachvererbung erbt von mehreren Basisklassen

Generalisierung Die spezialisierte Klasse erweitert die Liste der Attribute, Operationen und Assoziationen der Basisklasse

Generalisierungsmenge spezifiziert, nach welchen Kriterien eine Generalisierungstruktur erstellt wird. (z.B. nach Tätigkeit, Job, etc.)

2.9 Aktivität Balzert S. 69

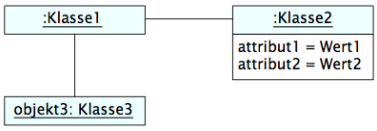
Eine Aktivität beschreibt die Ausführung von Funktionalität bzw. Verhalten.

2.10 Zustandsautomat Balzert S. 87

Zustandsdiagramm (finit state diagram)

2.11 Objektdiagramm (*object diagram*) Balzert S. 21

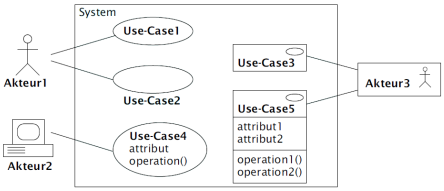
Momentaufnahme / Schnappschuss des Systems.
Beschreibt Objekte, Attributwerte, Objektbeziehungen zu einem bestimmten Zeitpunkt.



2.12 Use-Case Diagramm Balzert S. 62

Ein Use-Case spezifiziert eine Sequenz von Aktionen

Akteur ist eine Rolle, die ein Benutzer des Systems spielt. Jeder Akteur hat einen Einfluss auf das System. Befindet sich stets ausserhalb des Systems



Schablone Balzert S.68

Ziel:	globale Zielsetzung bei erfolgreicher Ausführung des Geschäftsprozesses.
Kategroie:	primär, sekundär, optional
Vorbedingung:	erwarteter Zustand, bevor der Use-Case beginnt.
Nachbedingung Erfolg:	erwarteter Zustand, nach erfolgreicher Ausführung des Use-Cases (Ergebnis)
Nachbedingung Fehlschlag:	erwarteter Zustand, wenn das Ziel nicht erreicht werden kann
Akteure:	Rollen von Personen oder anderen Systeme, die den Use-Case auslösen oder daran beteiligt sind.
Auslösendes Ereignis:	Wenn dieses Ereignis eintritt, dannn wird der Use-Case initiiert.
Beschreibung:	1. Erste Aktion 2. Zweite Aktion
Erweiterungen:	1a Erweiterung des Funktionsumfangs der ersten Aktion
Alternativen:	1a Alternative Ausführung der ersten Aktion 1b Weitere Alternativen zur ersten Aktion 2b ...

2.13 Szenario Balzert S. 80

Ein Szenario ist eine Sequenz von Verarbeitungsschritten, die unter bestimmten Bedingungen auszuführen sind.
Die Szenarien lassen sich in zwei Kategorien unterteilen, die die eine erfolgreiche Bearbeitung des Use-Case beschreiben und jene Szenarien welche zu einem Fehlschlag führen.

Diagrammübersicht:

- Sequenzdiagramm → Balzert S.81
- Kommunikationsdiagramm → Balzert S.85
- Timing-Diagramm
- Interaktionsübersichtsdiagram

2.14 Analyseprozess Balzert S. 130

Der Analyseprozess beschreibt die methodische Vorgehensweise zur Erstellung eines obejektorientierten Analysemodells.

Es besteht aus

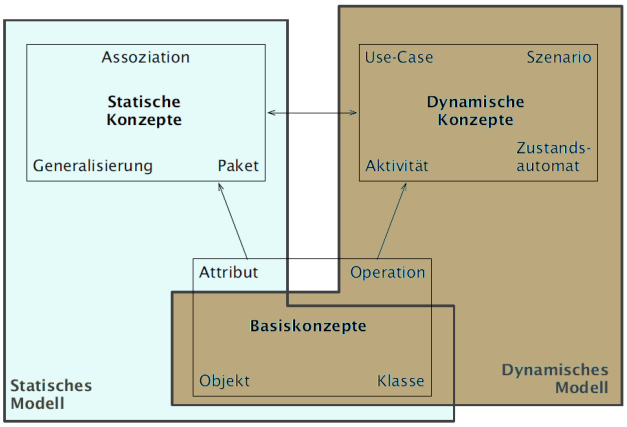
Makroprozess Der Makroprozess legt die methodischen Schritte fest, sprich in welcher Reihenfolge die Checklisten abgearbeitet werden → siehe Balzert S.132

Checklisten siehe Balzert S.133

3 OOD

3.1 Ziel des Entwurfs

- Spezifizierte Anwendung auf einer Plattform unter den gefordeten technischen Randbedinungen zu ralisieren
- Entwurf liegt noch in einem höheren Abstaktionsniveau als in der Implementierung
- In der Entwurfsphase wird das OOD-Modell unter den Gesichtspunkten der Effizienz und Standartisierung konzipiert.
- Jede entworfene Klasse kann direkt implementiert werden
- Das OOD-Modell baut auf dem OOA-Modell auf
- Die Namen in dem OOD-Modell sollten der Syntax der jeweiligen Progarmmiersprache entsprechen.



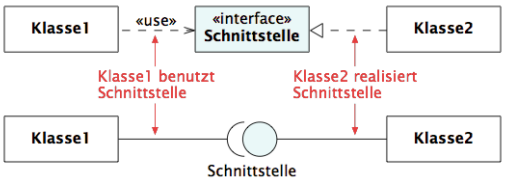
3.2 Klassen Balzert S. 252

parametrisierte Klasse Template

Container Klasse dient zur Verwaltung von einer Menge von Objekten einer anderen Klasse (typischerweise als Array implementiert)

Schnittstelle Balzert S.256

ähnlich wie abstrakte Klasse, aber **keine Operation** ist implementiert



Implementierung siehe Balzert S.258, 259

3.3 Attribut Balzert S. 261

Sichtbarkeit

\+ public
\# protected
\− privat
\~ package

Klassenattribute werden durch Unterstrichen gekennzeichnet und mit **static** implementiert

Notation Sichtbarkeit / **name** : Typ [Multiplizität] = Anfangswert {Eigenschaftswert}

Eigenschaftswerte siehe Balzert S.263

Implementation siehe Balzert S.264

Beispiel
anfangsbestand: Integer = 0 # /gesamtsumme: Currency - vornamen: String [1..3] {ordered} - kontonr: Integer {readOnly, key} - titel: String [0..1] + anzahl: Integer ~ person: Person

3.4 Operationen Balzert S. 266

Notation Sichtbarkeit **name** (Parameterliste) : Ergenistyp {Eigenschaftswert}

Parameter Richtung parametername : Typ [Multiplizität] = Anfangswert {Eigenschaftswert}
Richtung:
- in
- out
- inout
- return

siehe Balzert S.267

Eigenschaftswerte siehe Balzert S.268

Implementation siehe Balzert S.269

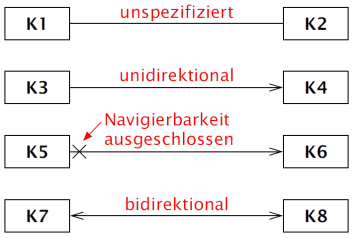
3.5 Komponente Balzert S. 273

Eine Komponente ist ein Softwarebaustein, der über klar definierte Schnittstellen, Verhalten bereitstellt. Das Innenleben der Komponente bleibt gegen aussen verborgen

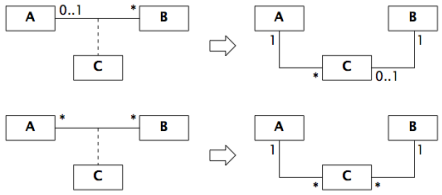
3.6 Assoziation Balzert S. 284

By-Value wird automatisch mit einer Komposition erstellt.

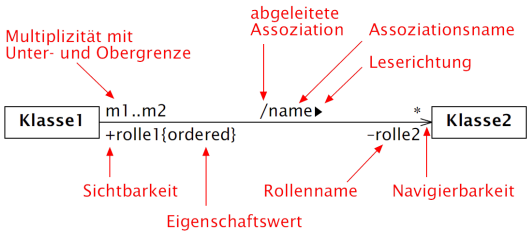
Navigierbarkeit

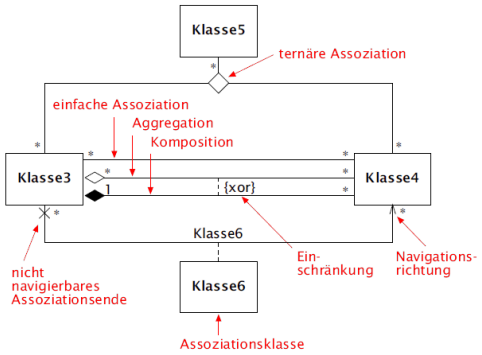


Auflösen einer Assoziationsklasse



Notation





Eigenschaftswerte siehe Balzert S.288

Implementation siehe Balzert S.291

3.7 Polymorphismus Balzert S. 293

Polymorphismus ermöglicht es, den gleichen Namen für gleichartige Operationen, in verschiedenen Klassen zu gebrauchen.

Notation Operation *kursiv* schreiben

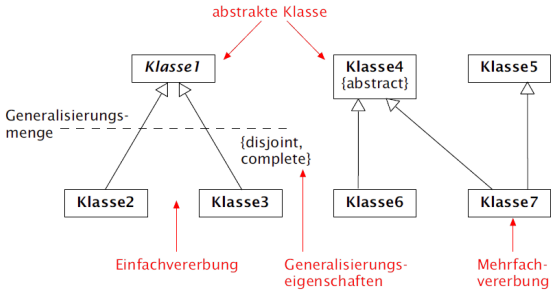
Dekleration mit **virtual** deklarieren

Implementation siehe Balzert S.298

3.8 Generalisierung Balzert S. 300

Generalisierungseigenschaften siehe Balzert S.302

Notation



Implementation siehe Balzert S.305

3.9 Szenario Balzert S. 327

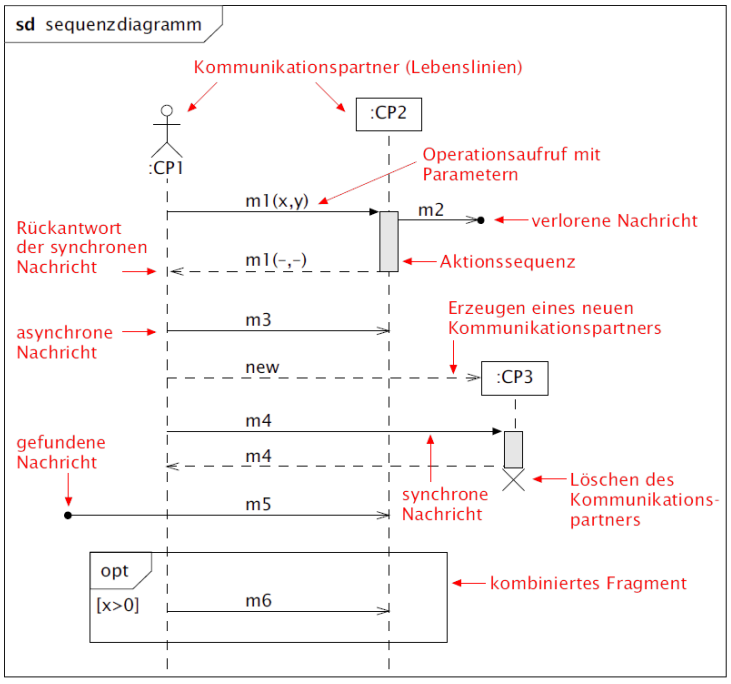
bezieht sich auf Sequenzdiagramme

synchrone Nachricht Der Sender wartet bis der Empfänger die Verarbeitung komplett durchgeführt hat.

asynchrone Nachricht Der Sender wartet **nicht** bis auf den Empfänger, sondern setzt seine Verarbeitung parallel fort.

kominierte Fragmente siehe Balzert S.331

Notation



3.10 Entwurfsmuster Balzert S. 355