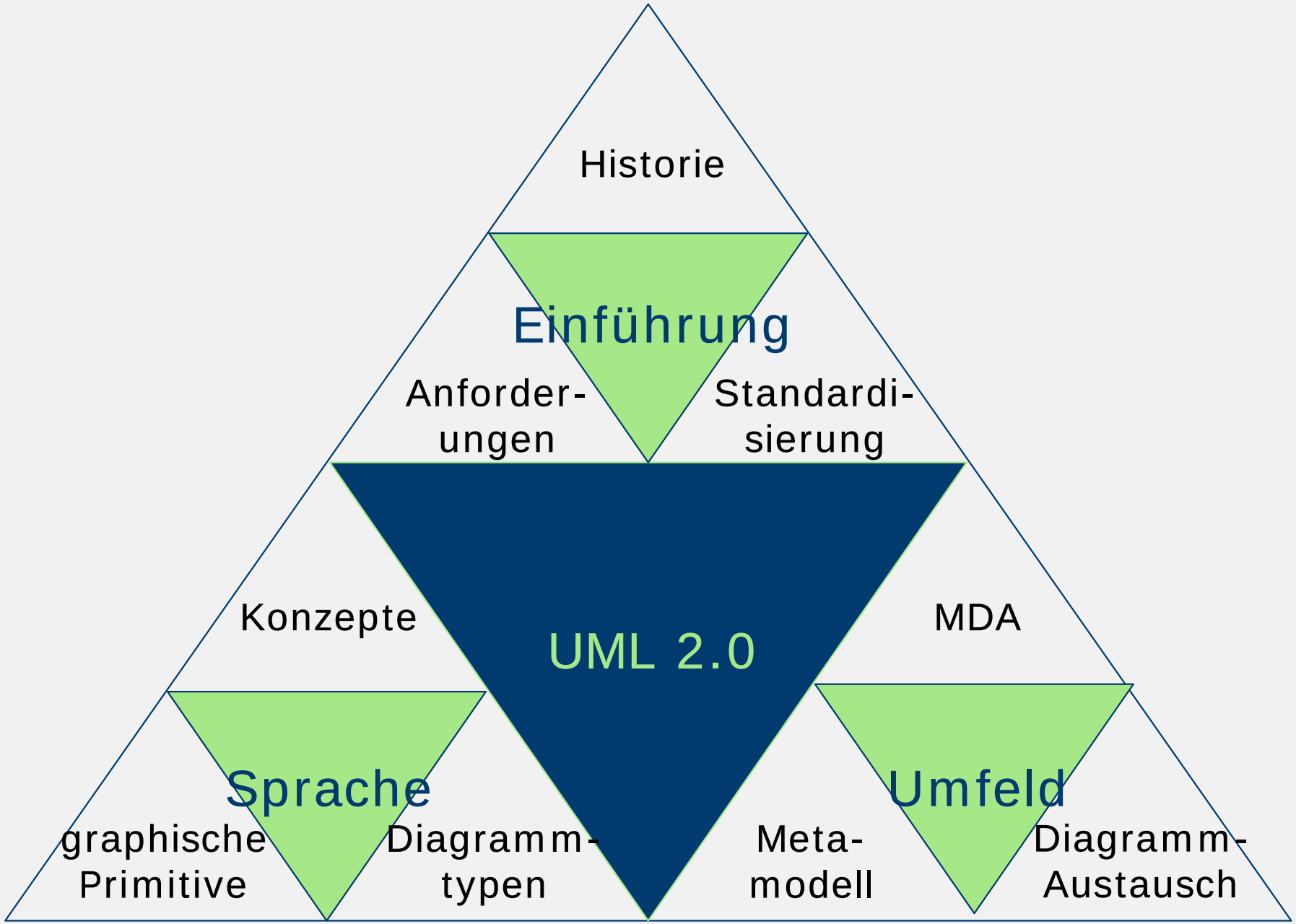
UML 2.0

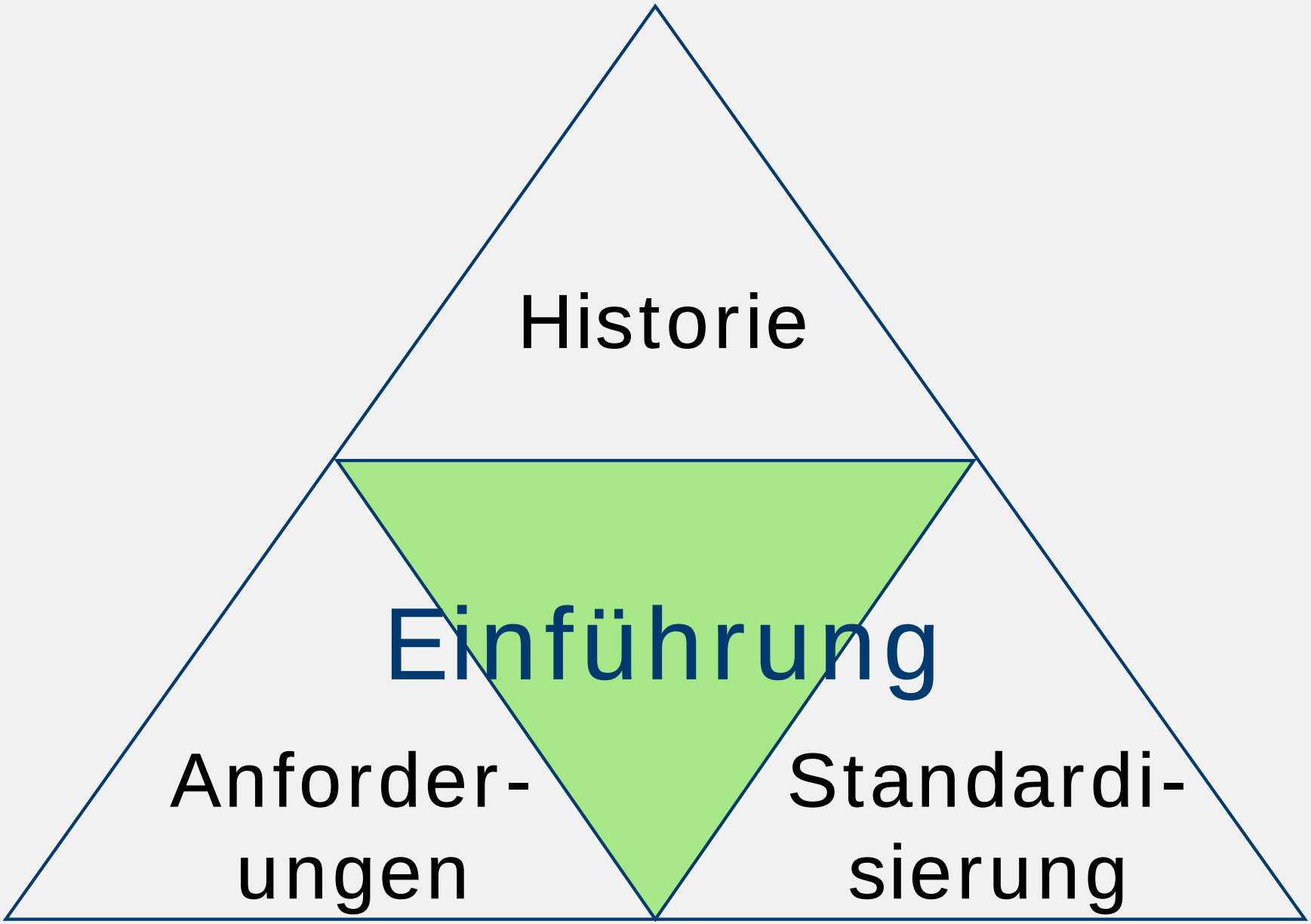
Die neue Version der Standardmodellierungssprache

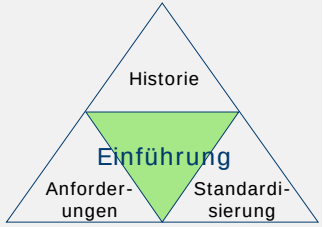
Prof. Mario Jeckle

Fachhochschule Furtwangen
mario@jeckle.de
http://www.jeckle.de

Modellierung 2004, Marburg, 2004-03-24


Historie
Einführung
Anforder-
ungen
Standardi-
sierung
Konzepte
UML 2.0
MDA
Sprache
Umfeld
graphische
Primitive
Diagramm-
typen
Meta-
modell
Diagramm-
Austausch


Historie
Einführung
Anforder-
ungen
Standardi-
sierung



Vom Methodenkrieg ...

OML
Firesmith, Henderson-Sellers, Page-Jones; 1998

Unified Modeling Language v1.0
Booch, Jacobson, Rumbaugh; 1997

Catalysis
D'Souza, Willes; 1996

Unified Method
Booch, Rumbaugh; 1995

MOSES
Henderson-Sellers; 1994

SOMA
Graham; 1994

Fusion
Coleman, et al.; 1994

OBA
Rubin; 1992

BON
Nerson; 1992

OOA&D
Martin, Odell; 1992

OOD
Booch; 1992

OOSE
Jacobson; 1992

OOAD&I
Henderson-Sellers, Macrone; 1992

OOA
Coad, Yourdan; 1991

OMT
Rumbaugh, et al.; 1991

OSA
Embley; 1991

OOSA
Shlear, Mellor; 1991

SCOOP
Cherry; 1990

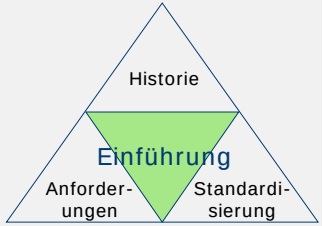
HOOD
ESA; 1990

RDD
Wirfs-Brock; 1990

OBA
Bailin; 1989

State Charts
Harel; 1987

... zum Standardisierungskrieg



Erweiterung

OMG Unified Modeling Language 2.0
UML 2 Partners; 2003+

Breiteneinsatz

Erfahrungen
der Anwender

OMG Unified Modeling Language 1.5
UML Partners; 2003

OMG Unified Modeling Language 1.4
UML Partners; 2001

Object Management Group
übernimmt Copyright

Standardisierung

OMG Unified Modeling Language 1.3
UML Partners; 1999

XML Metadata
Interchange

(OMG Unified Modeling Language 1.2
UML Partners; 1998)

Unified Modeling Language 1.1
UML Partners; 9/1997

Vereinheitlichung

Integration der
Object Constraint Language

Unified Modeling Language 1.0
UML Partners 1/1997

Einsatzerfahrung
der Sprachschöpfer

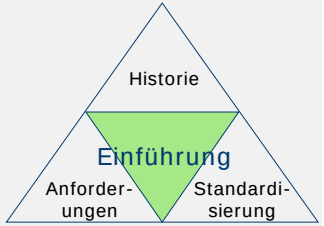
Unified Modeling Language 0.9, 0.91
Booch, Rumbaugh, Jacobson; 1996

Unified Method 0.8
Booch, Rumbaugh 1995

OOSE
Jacobson; 1992

...

UML-Anwender	100.. / 1..1000	UML-Entwickler
UML-Anwender	50.. / 1..100	UML-Entwickler
UML-Anwender	3.. / 1..	UML-Entwickler



Der Weg zur UML 2

Microsoft
Hewlett-Packard
Oracle
Sterling Software
MCI Systemhouse
Unisys
ICON Computing
Rational Software

IntelliCorp
i-Logix
IBM
ObjecTime
Platinum Technology
Ptech
Taskon
Reich Technologies
Softeam

Ericsson
IONA
Motorola
Telelogic
Boldsoft
Adaptive
Financial System Architects
SUN
Gentleware
DaimlerChrylser

Alcatel
Kabira
Kennedy Carter

Computer Associates

0.9

1.0

1.1/2

1.3

1.4

1.5

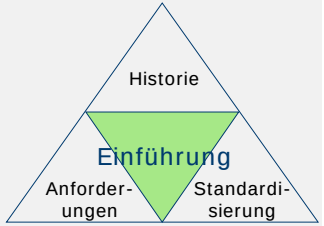
2.0

OMG
EDS

Sterling Software
ICON Computing
ObjecTime
Platinum Technology

Microsoft
Hewlett-Packard
ICON Computing
IntelliCorp
Ptech
Taskon
Reich Technologies

MCI Systemhouse

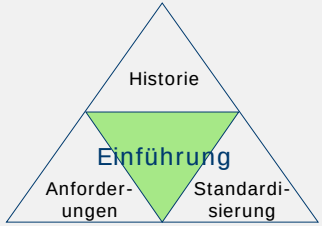


Warum eine neue Version?

- **Evolution**
  - Der Markt hat sich bewegt...
    - Neue Programmiersprachen (z.B. C#, Python, PHP)
    - Neue Anwendungsdomänen
      (z.B. Serverprogrammierung, Echtzeitanwendungen)

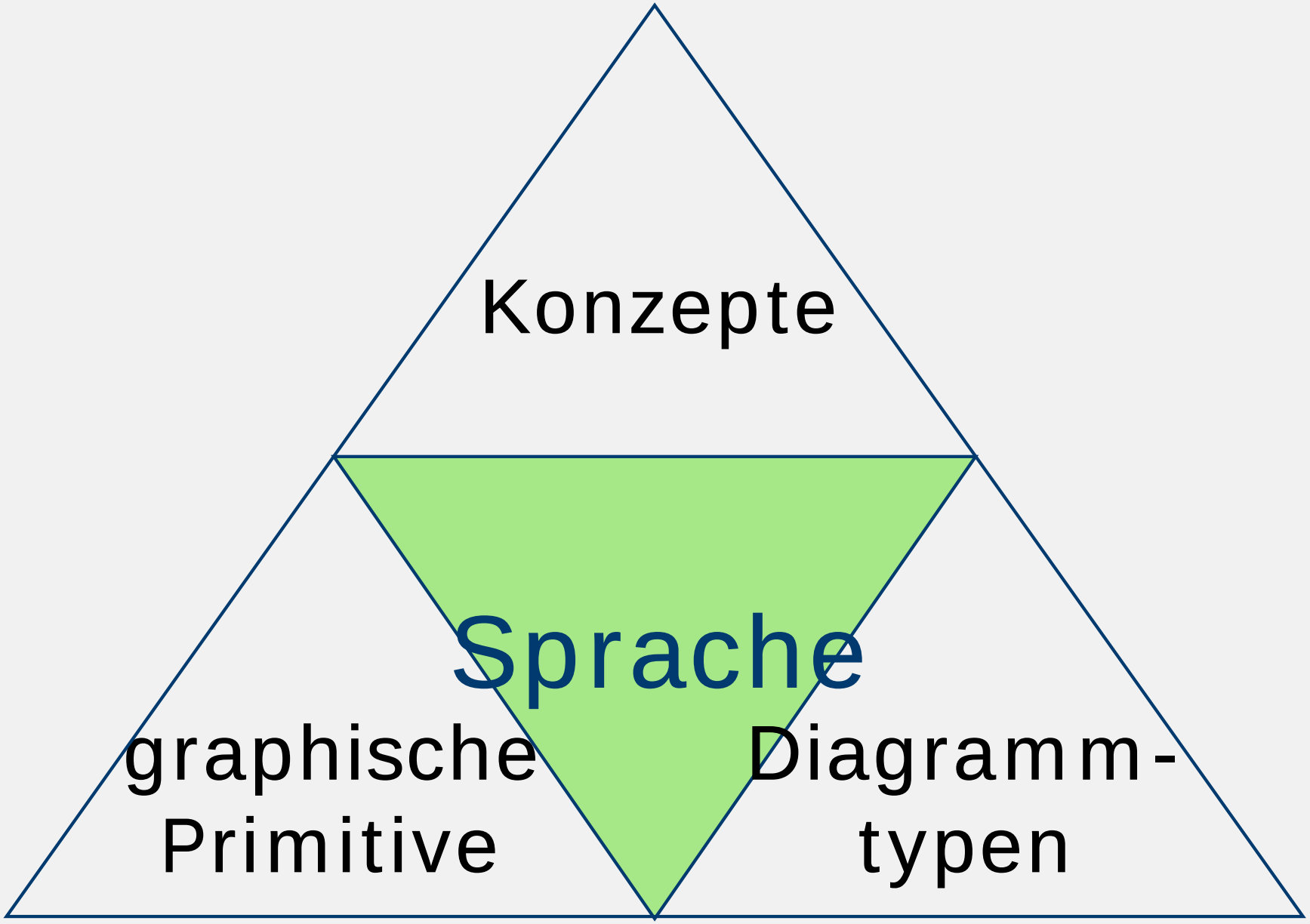
- **Erfahrung**
  - Für einige Einsatzgebiete bietet UML v1.x ...
    - Manchmal zu wenig Konstrukte
    - Manchmal zu viele
    - Machmal so viele, dass die sinnvolle Auswahl schwerfällt

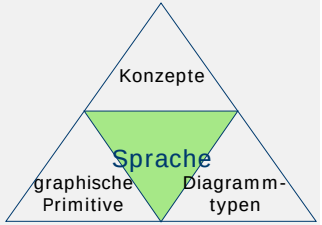
- **Eliminierung**
  - Einige Programmiersprachen verschwinden (z. B. C++)
  - Einige früher als modellierungsnah eingestufte Konzepte
  - entwickeln sich inzwischen getrennt von UML weiter
  - (z. B. Entwicklungsprozesse, Codegenerierung)



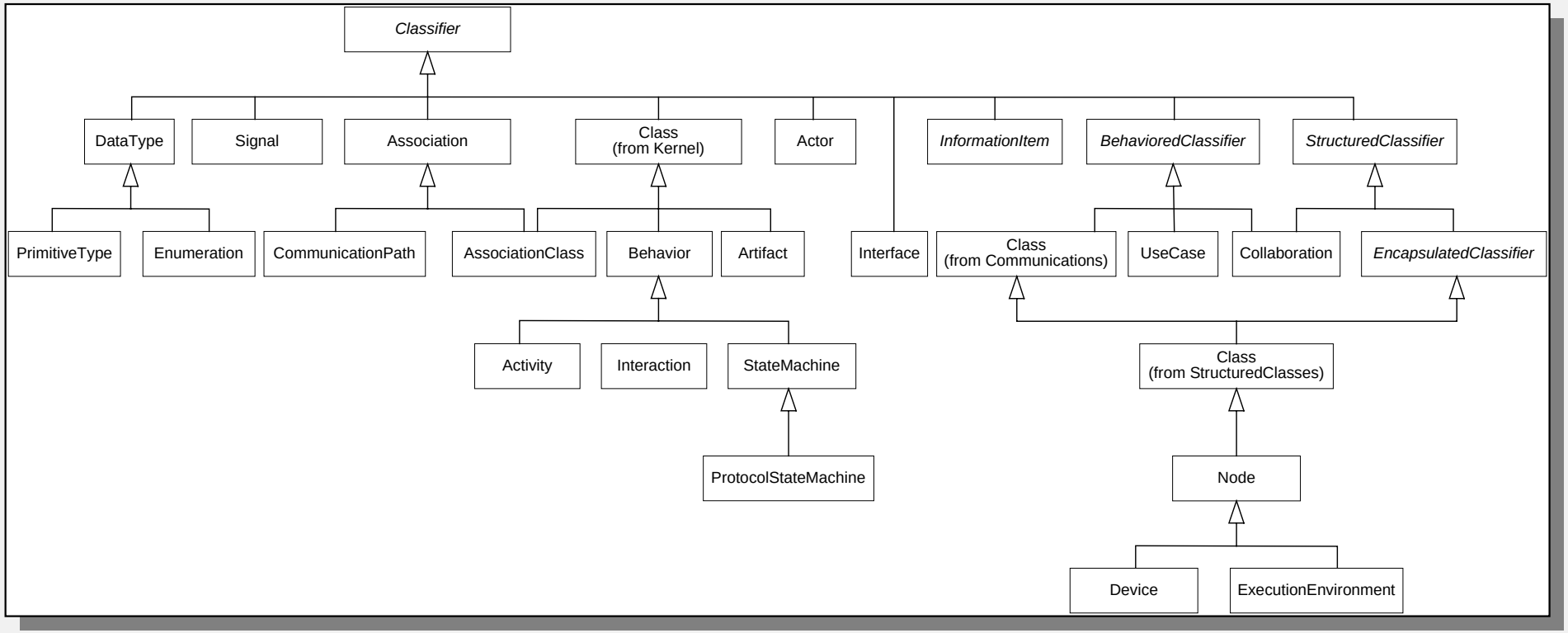
Ziele der UML 2

- **Übersichtlichkeit**
  - Weniger graphische Modellkonstrukte
  - Weniger Basiskonzepte
  - Wiederverwendung von Basiskonzepten
- **Präzisionssteigerung**
  - Reformulierung des Meta-Modells
  - Weitestgehende OCL-Verwendung
  - Unveränderte Wiederverwendung von Basiskonstrukten soweit sinnvoll möglich
- **Ausführbarkeit**
  - Erweiterte Zustandsmaschinen
  - Stärkere Beziehungen zwischen statischen und dynamischen Diagrammen
  - Integration erprobter Konzepte außerhalb der UML


Konzepte
Sprache
graphische Primitive
Diagramm-typen

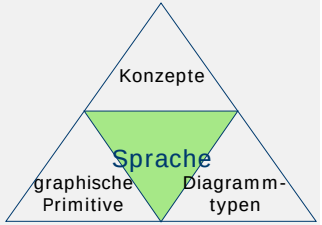


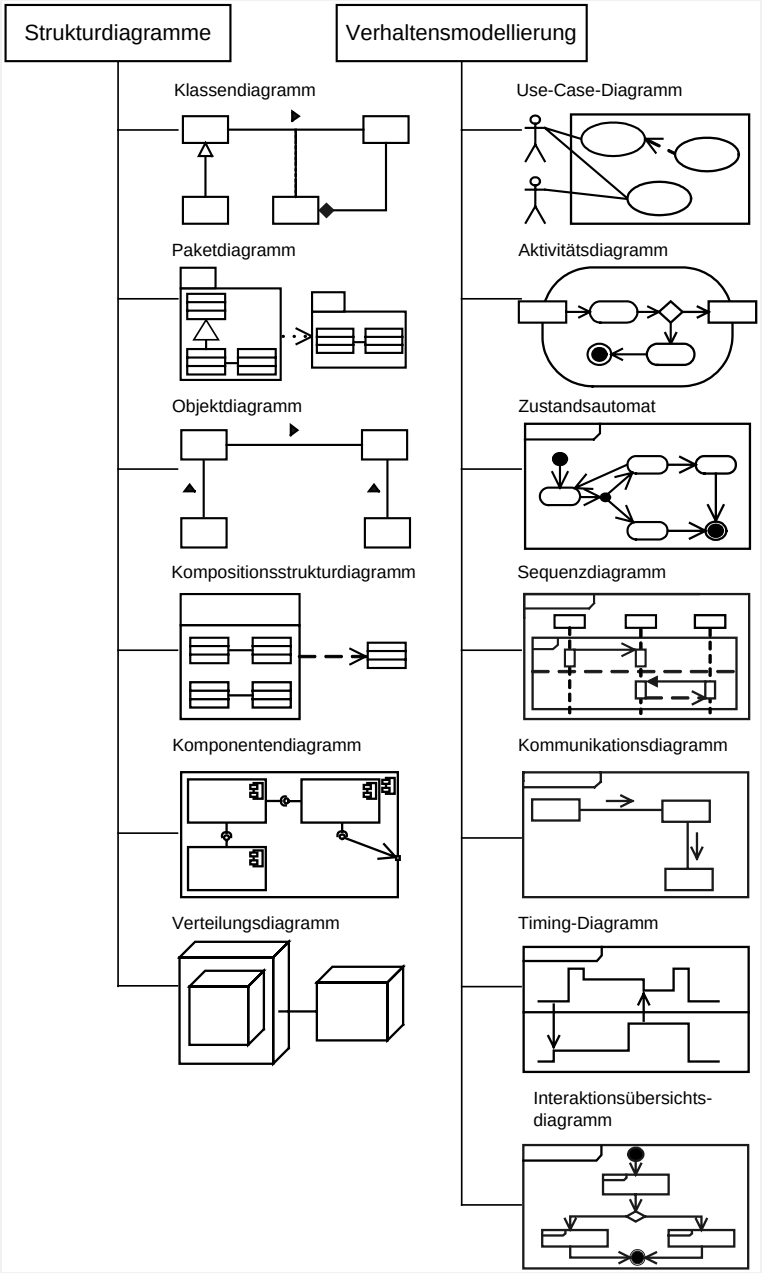
Classifier-Konzept



- Erweiterte Nutzung der Classifier:
- … können durch Beziehungen (Assoziationen) verknüpft werden
- … können Charakteristika (Attribute) besitzen
- … können Verhaltensspezifikationen (Operationen) besitzen
- … können generalisiert werden
- … können autonom auf Signale reagieren
- … können ausschließlich der Strukturierung dienen (abstract)

Die Diagrammsprachen



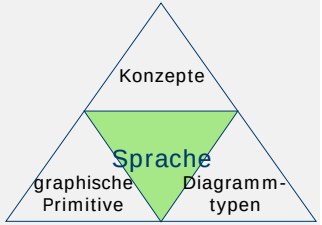


Nur marginale Änderungen
- Klassendiagramm
- Use-Case-Diagramm
- Objektdiagramm

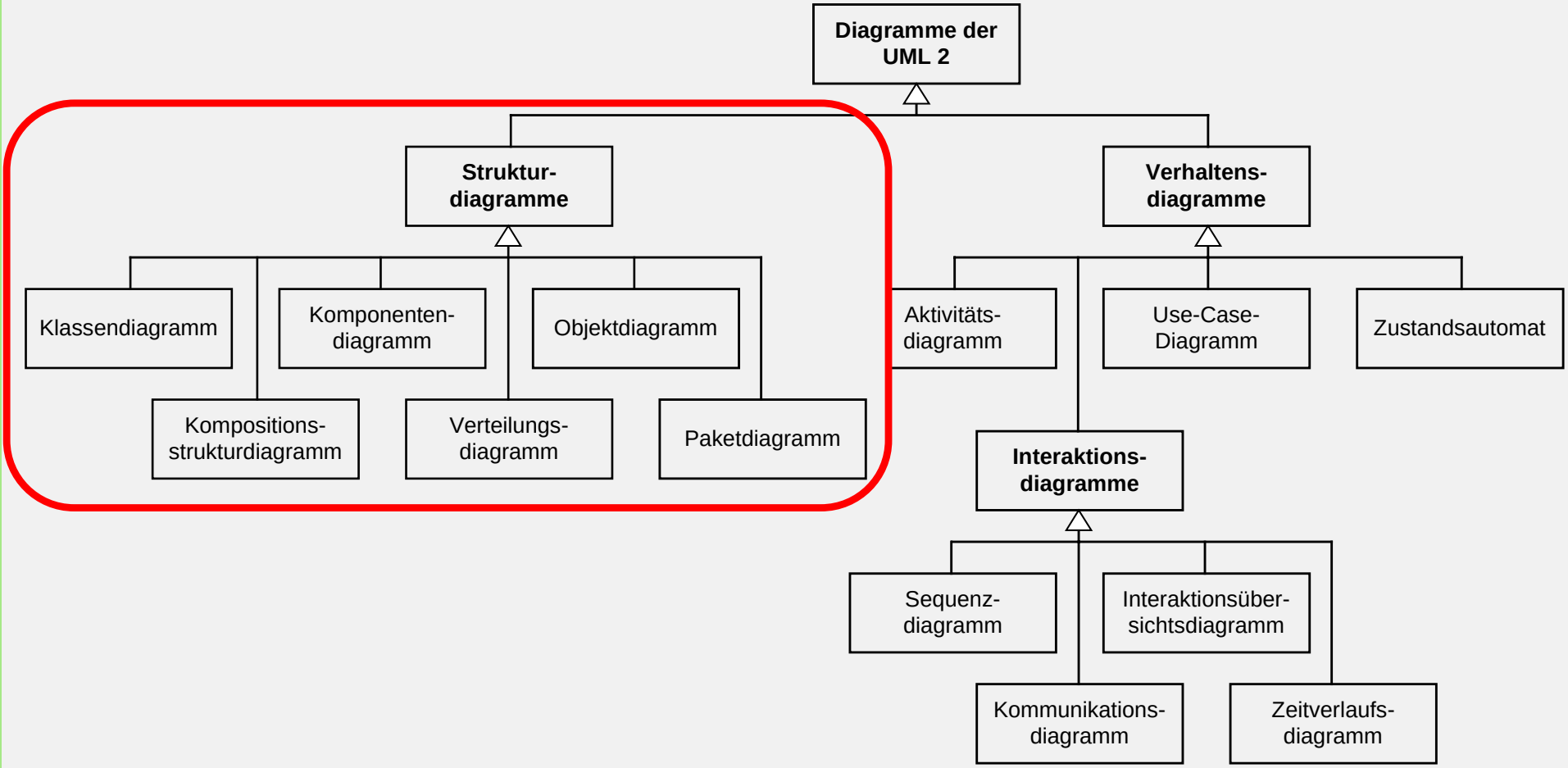
Klein(-e Änderungen) aber oho:
- Paketdiagramm
- Verteilungsdiagramm

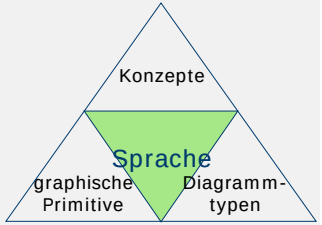
Massiv und tiefgreifend verändert:
- Aktivitätsdiagramm
- Sequenzdiagramm
- Zustandsautomat

Vollständig neu:
- Kompositionsstrukturdiagramm
- Interaktionsübersichtsdiagramm
- Timing-Diagramm
- Kommunikationsdiagramm



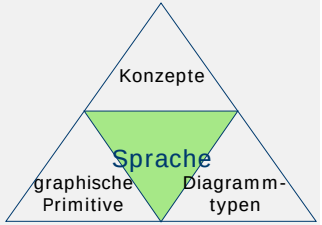
Die Strukturdiagramme





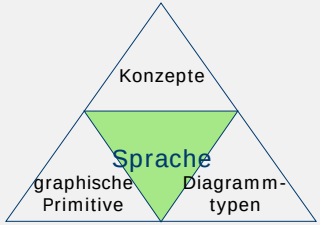
Die Strukturdiagramme

- **Klassendiagramm**
  Eine Zusammenstellung deklarativ-statischer Modellelemente (d.h. Klassen, Typen, ihre Inhalte und Beziehungen)
- **Objektdiagramm**
  Enthält Objekte und Beziehungsausprägungen
- **Paketdiagramm**
  Stellt die logische Organisation von Modellelementen und deren Abhängigkeiten dar
- **Komponentendiagramm**
  Zeigt die Organisation und Abhängigkeiten von Komponenten
- **Kompositionsstrukturdiagramm**
  Zeigt die interne Struktur eines Classifiers sowie seine Möglichkeiten zu Interaktion mit anderen Systemkomponenten
- **Verteilungsdiagramm**
  Zeigt die Ausführungssicht des Systems



Die Strukturdiagramme

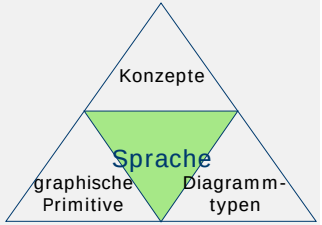
- Hintergrund
  - Statische Modellierung seit Mitte der 1970er Jahre gängig und hinreichend erforscht (z.B. Entity-Relationship)
  - Darstellung nicht ausführbarer statischer Zusammenhänge durch die ersten OO-Modellierungssprachen (u.a. Boochs OOSE, Rumbaughs OMT) weidlich bearbeitet
  - Strukturdiagramme (insbesondere Klassendiagramme) ausgereiftester und stabilster Teil der UML v1.x
- Jüngere Entwicklungstrends, die eine Neufassung rechtfertigen
  - Metamodellierung
  - Konzeptionelle Bereinigung
  - Wiederverwendung von bestehenden Konstrukten



Die Strukturdiagramme

- Statische Diagramme besitzen in der UML und ihrer Anwendung eine hervorgehobene Bedeutung
  - Bekanntester Diagrammtyp
  - Meistverwendester Diagrammtyp
  - Basis des UML-Metamodells
  - Basis des UML-Metametamodells (UML 2 Infrastructure Library bzw. Meta Object Facility)

- Änderungen
  - „automatisch“ sehr sichtbar
  - für den UML-Anwender „spürbar“
  - beeinflussen u.U. gesamtes Sprachkonzept
  - wirken sich auf andere (Meta-)Sprachen aus
  - Inhärente Bedeutung für den Modellaustausch zwischen Werkzeugen (XMI-Format wird aus Metamodell generiert)

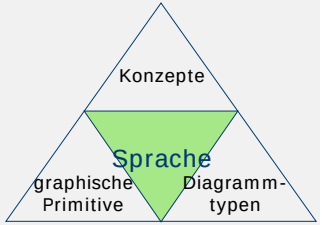


Klassendiagramm

- **Aufgabe**:
  - Zusammenstellung deklarativ-statischer Modellelemente (d.h. Klassen, Typen, ihre Inhalte und Beziehungen)
  - Definition der Sprachbasiskonzepte (Zur Erinnerung: Metamodell der UML ist als Klassendiagramm formuliert)

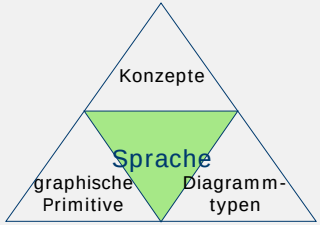
- **Aussage**:
  - Details über Daten- und Verhaltensstruktur des Systems

- **Aufgabe im Projekt:**
  - Variierend ...
  - von der ersten Darstellung konzeptueller Dateninhalte
  - über plattformunabhängige logische Modelle
  - bis hin zu „Implementierungsbauplänen“ („Bilder-für-Java“, „Kästchen-und-Strichchen-statt-C++“)
  - Ersetzt oftmals das klassische, in Entity Relationship-Notation abgefasste, Datenmodell



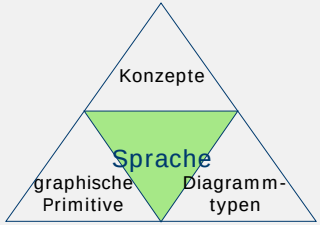
Klassendiagramm

- **Änderungen durch UML 2**:
  - Insgesamt: Marginalien!
  - Im Detail ...
    - Multiplizität nur noch genau ein Intervall
    - Attribute sind nicht mehr Kompositionsaggregation
    - Ordnung von Attributen und Operationen
    - Präferenz für Stereotypendarstellung geändert
    - *Diskriminator* entfällt, stattdessen wird Generalisierungspfeil mit Namen annotiert
    - `realization`-Abhängigkeit zwischen Komponenten zugelassen
    - Stereotypen vereinheitlicht (z.B. frozen vs. readOnly)
    - Innere Klassen sind generell Kompositionen
    - Parameterbindung in parametrisierten Klassen
      - durch *spitze* Klammern
      - durch Pfeilnotation (statt Liste)
    - ...



Klassendiagramm

Klassen-name
Klasse
abstrakte Klasse
Klasse1
Klasse2
Attribut-bereich
Operations-bereich
Rolle
Assoziations-name
Rolle1
Assoziation1
i..j
Assoziation
Multiplizität
Abhängigkeits-beziehung
Generalisierung
Kompositon
Klasse3
{Eigenschaft1}
Eigenschaft
Aggregation
Schnittstelle
«interface»
Schnittstelle1
Stereotyp
Implementierte Schnittstelle
Benötigte Schnittstelle



Klassendiagramm

Feier
+termin : Datum
abbrechen()

unterhält

besucht *

Partyteilnehmer
/betrunken : Boolean = false
~intus : Cocktail [1..*]
#begeistert : Begeisterung
#überdenkeBegeisterung()
~trinke(in c : Cocktail)

Zutat
+name : String

Häppchen

0..*
isst {ordered}

«interface»
Esser
iss(in h : Häppchen; return satt:Boolean)

Cocktail
Zutaten : Zutat [1..*]

1..*
mixt

Barmixer
+mixe(in z : Zutat [1..*], out c:Cocktail)

1
mixt für

Esser

Esser

Gast
feiere()

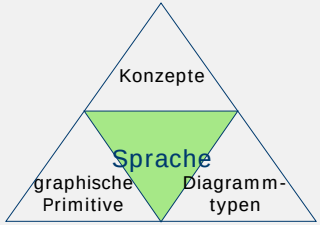
1..*
verabschiedet
{unique}

1..*
empfängt
{unique}

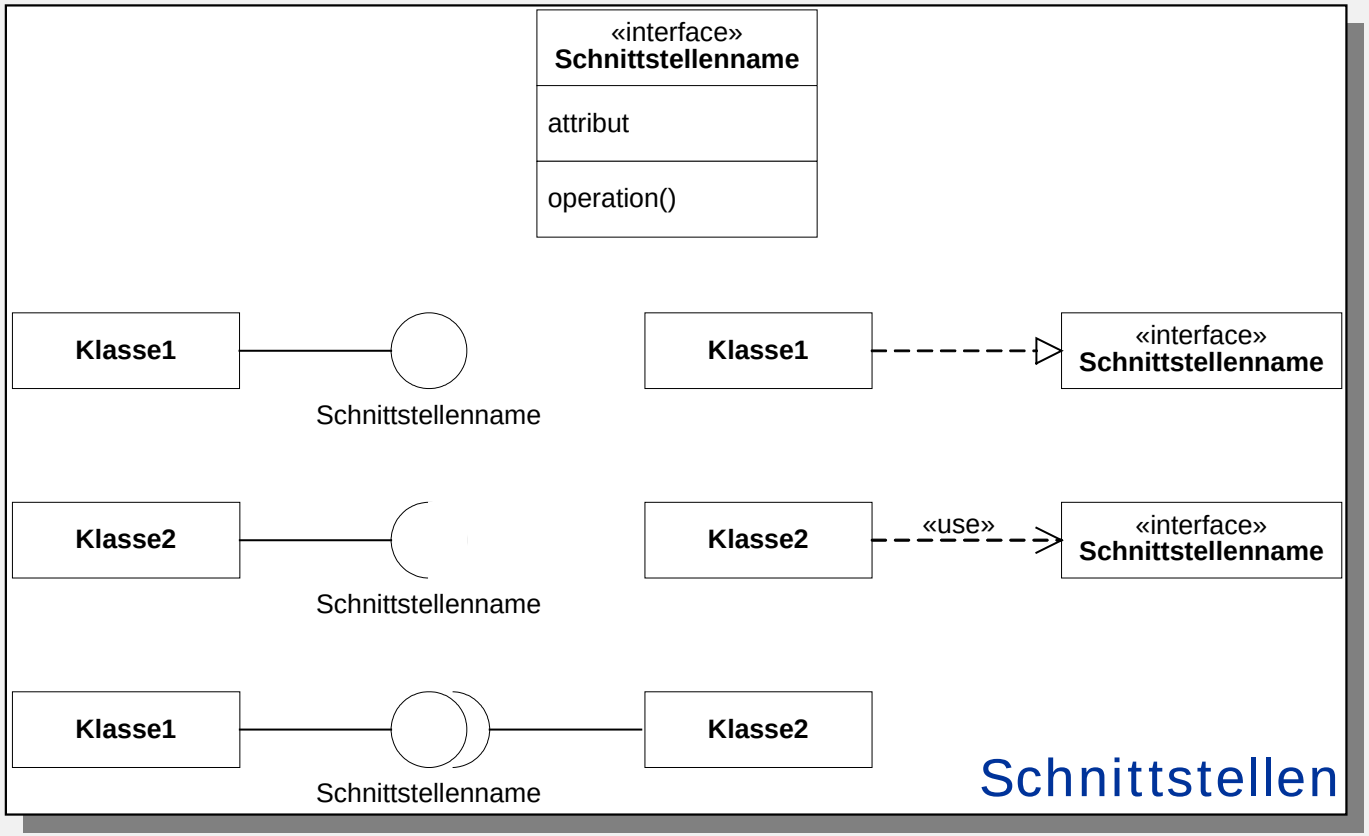
Gastgeber
hektisch : Boolean = true
begruesse(in g : Gast)
verabschiede (in g : Gast)
treibeAn(in b : Barmixer)

treibt an
1..1

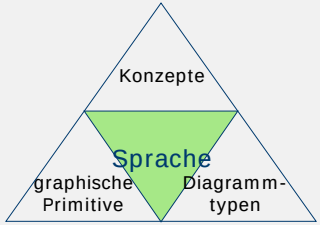
«enumeration»
Begeisterung
ekstatisch
verzückt
neutral
gelangweilt
überdrüssig



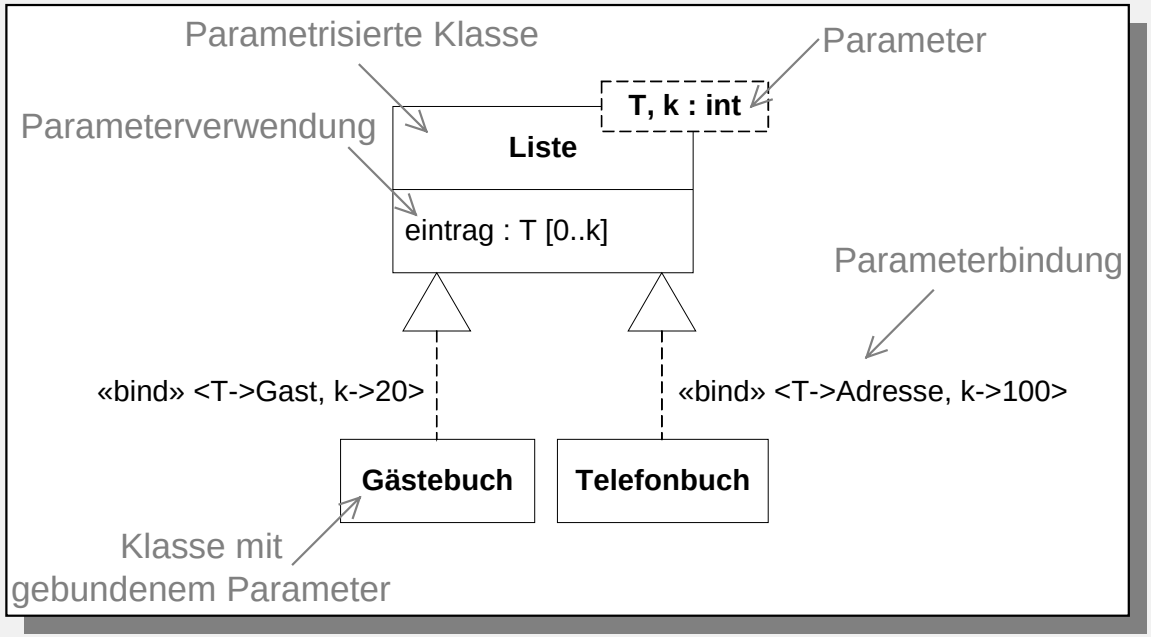
Klassendiagramm



- Ausschließlich durch Attribute und Operationen charakterisiert.
- Alle Charakteristika müssen öffentlich sichtbar sein.
- Häufigste graphische Darstellungsform: „Lollipopnotation“
- Eine Klasse kann mehrere Schnittstellen implementieren.
- Mit Schnittstellen lassen sich Teilaspekte der Vererbungssemantik (Typkonformität und Signaturvererbung) nachbilden.
- Eine Klasse, die eine Schnittstelle implementiert, muß jede darin vorgesehene Operation durch eine Methode manifestieren.

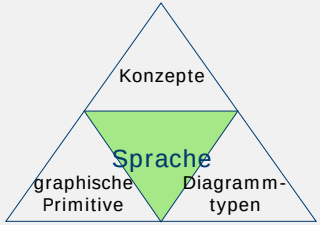


Klassendiagramm

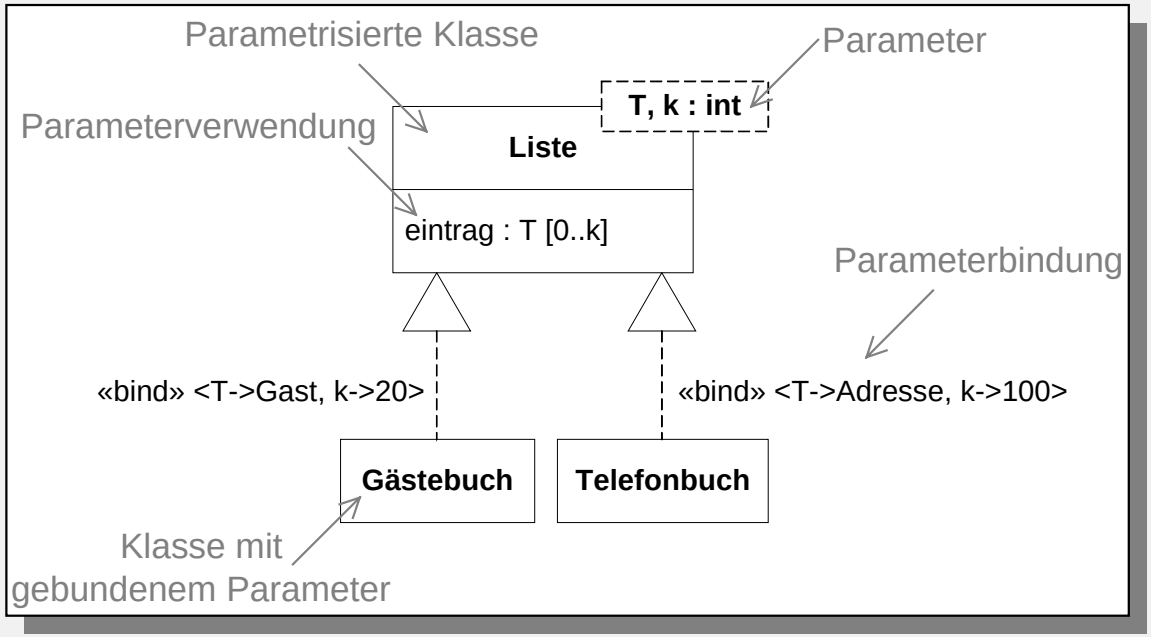


C++:

```
template <class T, int k>
class Liste {
        T elements[k];
};
...
Liste<Adresse,100> Adressliste;
Liste<Gast,20> Gaestebuch;
```

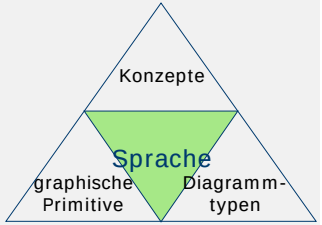


Klassendiagramm



Java:

```
class Liste<T> {
        Vector<T> elements;
        public Liste(int k) {
                elements = new Vector<T>(k);
        }
}
...
Liste<Adresse> Adressliste=new Liste<Adresse>(100);
Liste<Gast> Gästebuch=new Liste<Gast>(20);
```



Klassendiagramm

Superklasse

Partyteilnehmer

/betrunken : Boolean = false
~intus : Cocktail [1..*]
#begeistert : Begeisterung

#überdenkeBegeisterung()
~trinke(in c : Cocktail)

Partyveranstalter

+budget : Euro

+begleicheRechnung(betrag : Euro)

Generalisierung

Separat genutzt Generalisierung

Sub- und Superklasse

Gast

feiere()

Gastgeber

hektisch : Boolean = true

begruesse(in g : Gast)
verabschiede (in g : Gast)
treibeAn(in b : Barmixer)

begrüßt

Generalisierungs-menge

Teilnehmerstatus
{complete, disjoint}

Generalisierungs-eigenschaften

Gemeinsam genutzte Generalisierung

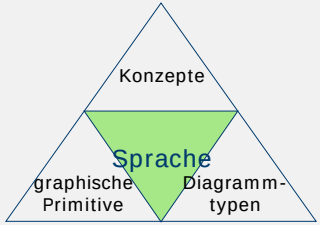
Partition

Partyneuling

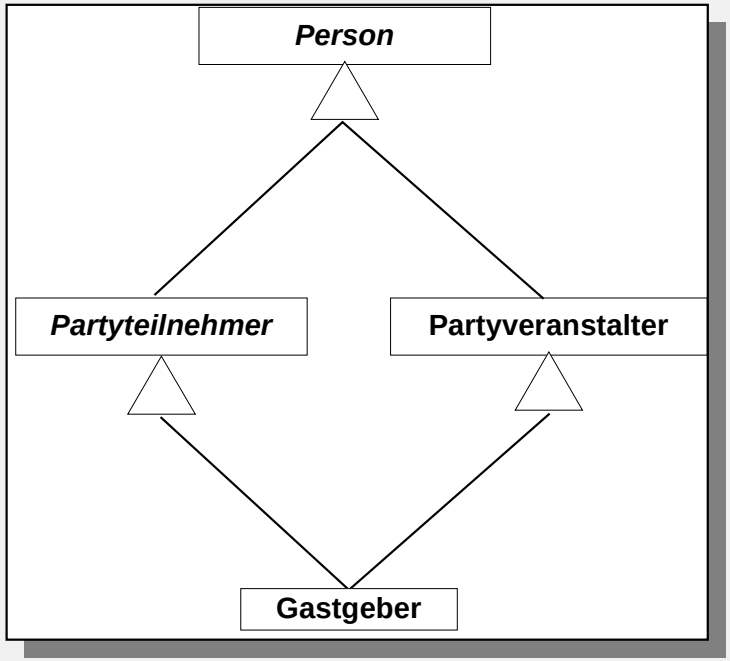
+vorstellen(g : Gast)

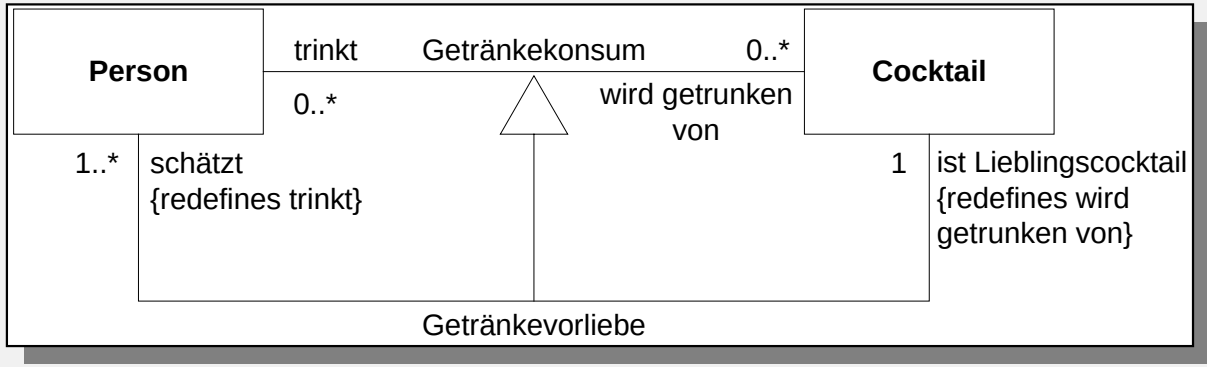
Stammgast

+selbstbedienungAmKühlschrank(c : Cocktail)

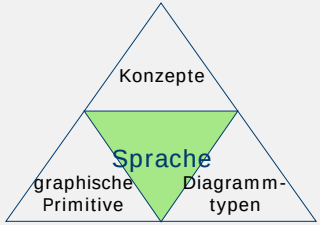


Klassendiagramm





- UML läßt generell die Angabe mehrerer Superklassen zu.
- In einigen (inzwischen fast der Mehrheit) Programmiersprachen ist die Anzahl der Superklassen auf genau eine beschränkt.
- Ausweg: Anwendbarkeit von Schnittstellen prüfen.
- Generalisierung von Assoziationen in UML zugelassen.
- In bekannten Programmiersprachen jedoch problematisch, da „Assoziation“ kein natives Sprachkonstrukt.
- Daher oft Darstellung der Assoziation durch eigenständige Klasse und Generalisierung dieser.



Klassendiagramm

Assoziationsname

Assoziation$_1$

Einschränkung

binäre Assoziation

Assoziation$_1$

{...}

nicht navigierbares Assoziationsende

Klasse$_1$

Klasse$_2$

Komposition

Navigationsrichtung

unspezifizierte Navigationsrichtung

Rolle

{Eigenschaft}

Aggregation

Klasse$_3$

Rolle

1..*

1..*

Eigenschaft

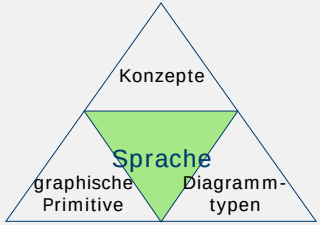
1..*

Multiplizität

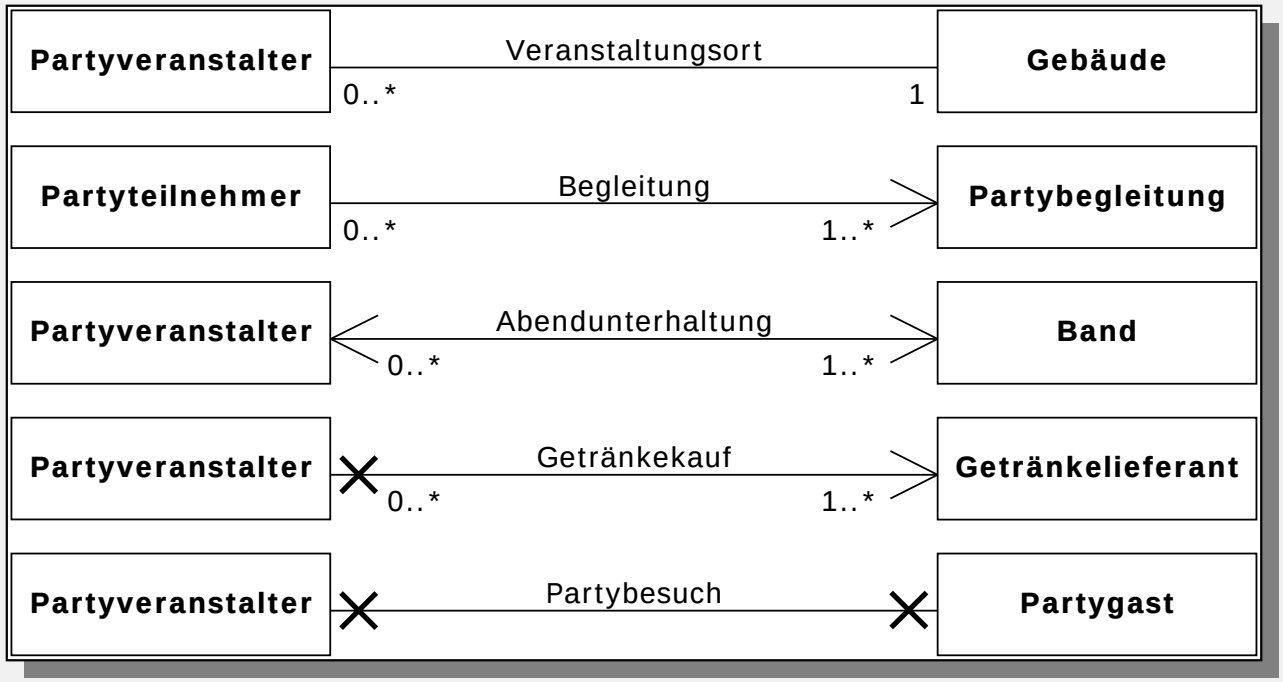
Rolle

n-äre Assoziation (ternäre Assoziation)

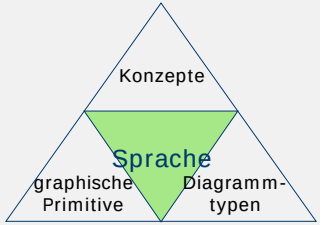
Assoziation$_2$



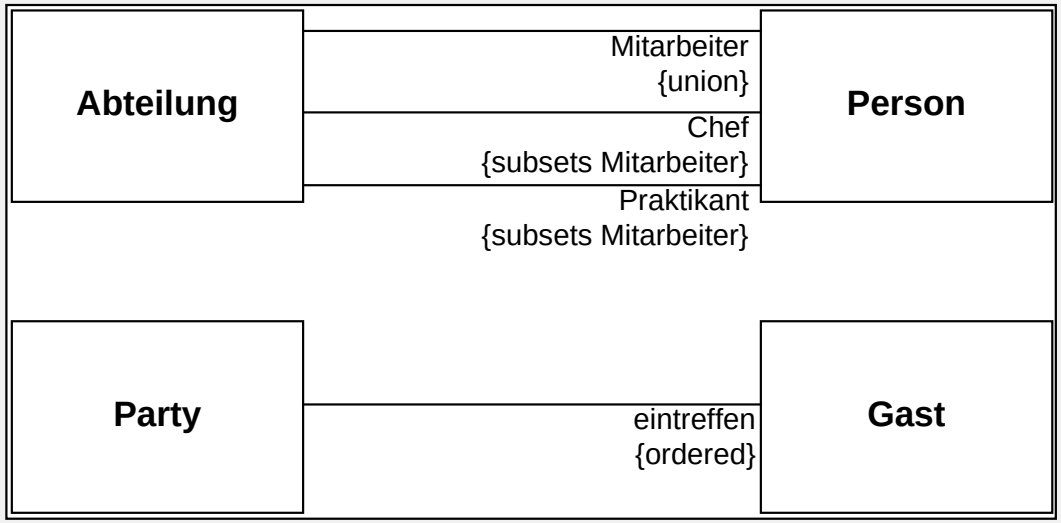
Klassendiagramm



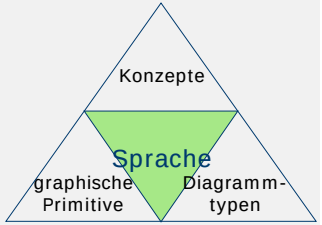
- Generell sind Assoziationen (in UML 2) mit unspezifizierter Navigabilität ausgestattet.
- In Programmiersprachen wird typischerweise ausschließlich unidirektionale Navigierbarkeit unterstützt
- Assoziationen mit beidseitigem Navigationsverbot praktisch kaum sinnvoll einsetzbar.



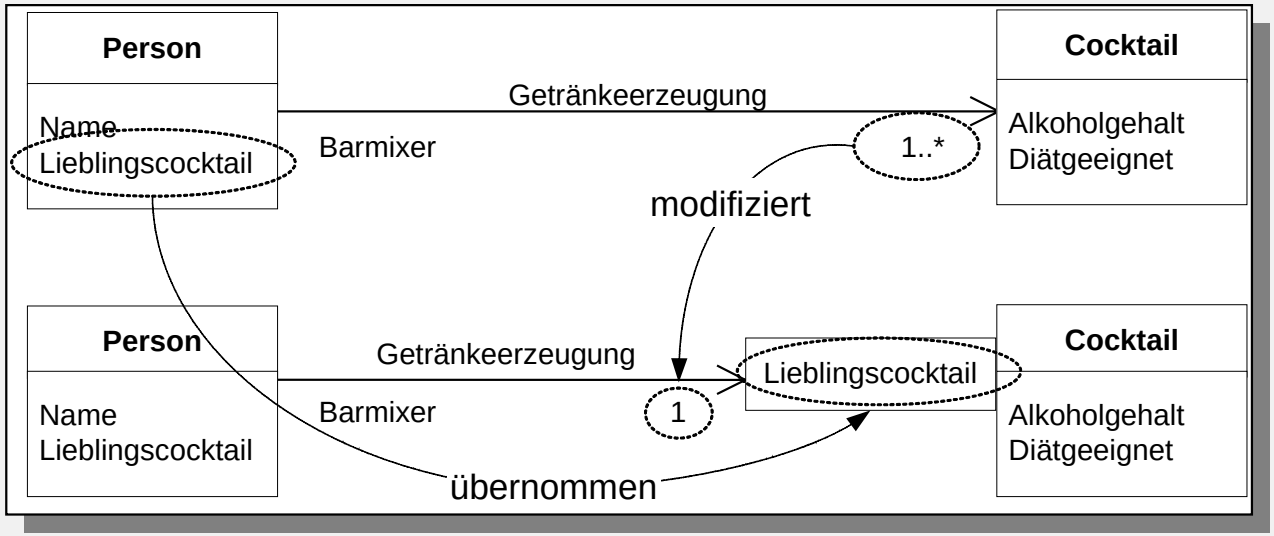
Klassendiagramm



- Beschränkung von Assoziationsenden aus semantischer Sicht sinnvoll.
- Keine vorgegebene Abbildung in Programmiersprache möglich.

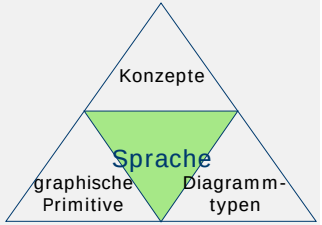


Klassendiagramm

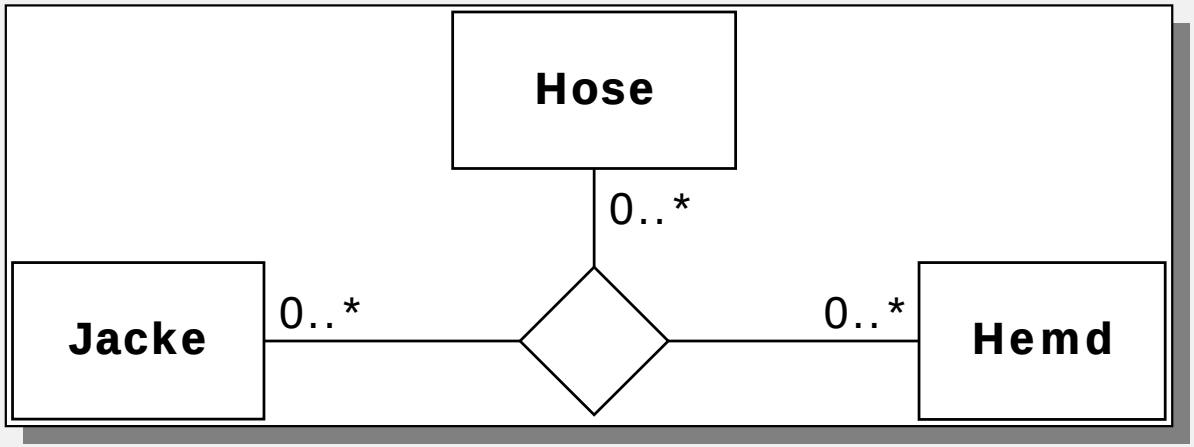


- Anmerkung zum Beispiel: Die beiden Darstellungen sind nicht äquivalent, sondern die untere Fassung stellt eine Restriktion der oberen dar!

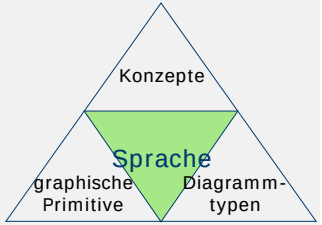
- Qualifizierte Assoziation zumeist im relationalen Datenbankumfeld anzutreffen.
- Modifiziert Assoziationsmultiplizität einschränkend.
- Objektorientiertes Analogon des relationalen Fremdschlüssels.



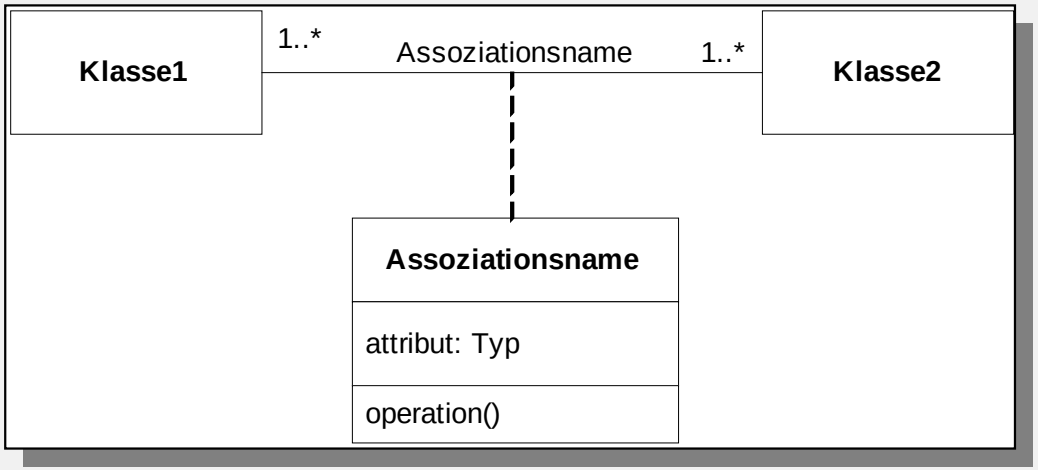
Klassendiagramm



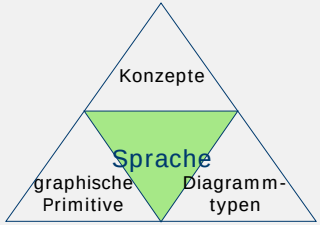
- *n*-äre Assoziation prinzipiell zugelassen.
- Vorsicht! Veränderte Interpretation der Multiplizität
  Sie bezieht sich nun auf einen Tupel, gebildet aus Ausprägungen aller durch die Assoziation verbundenen Klassen.
- In gängigen Programmiersprachen nicht direkt umsetzbar.



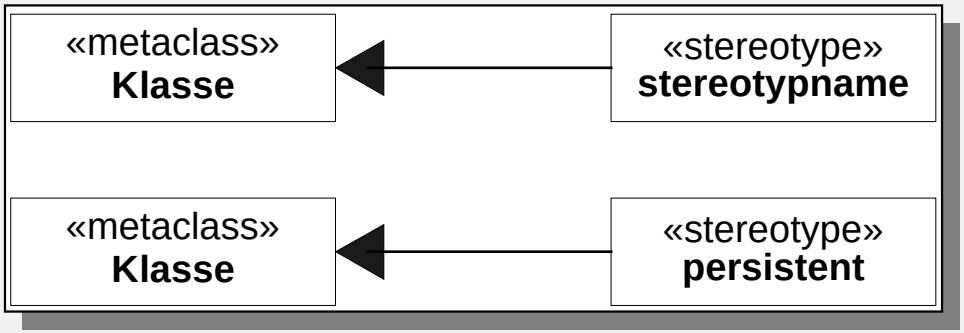
Klassendiagramm



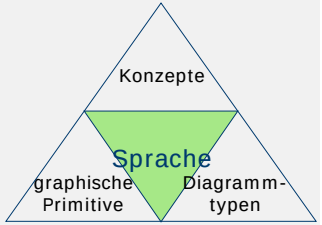
- Assoziationsklasse vereinigt Semantik der Klasse und der Assoziation, d.h. nicht nur „attributierte Assoziation“.
- Assoziationsklasse definiert Eigenschaften und Verhalten, welches konkreten Beziehungen (damit Instanztupeln) zugeordnet ist.
- Direkte Umsetzung in Programmiersprachen kaum möglich. Zumeist: Nachbildung durch „echte“ Klasse und zusätzliche Assoziationen.



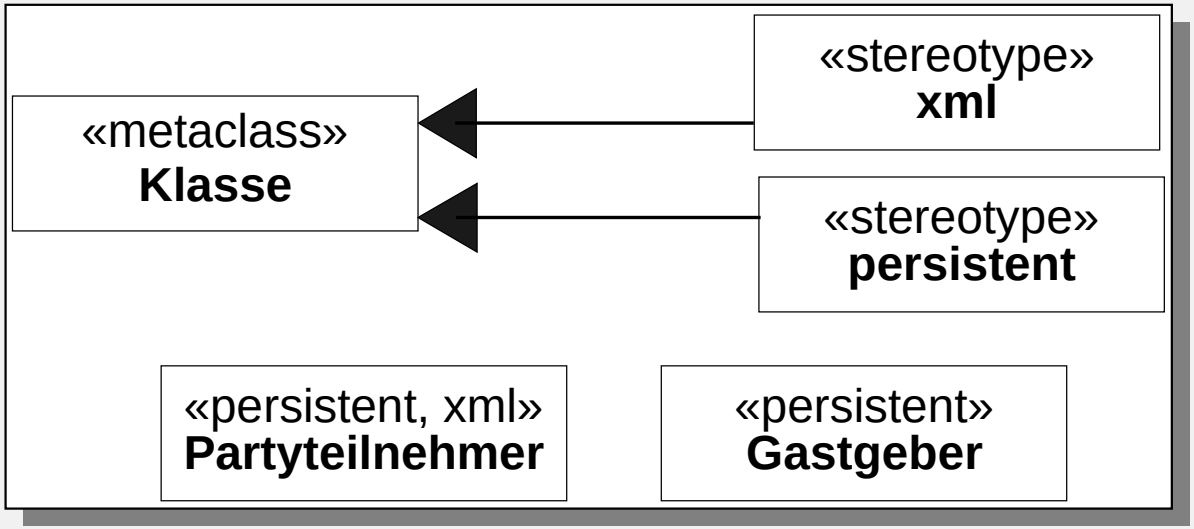
Klassendiagramm



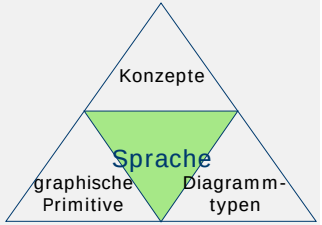
- Manchmal ist die in der Spezifikation vorgesehene UML nicht genug ...
- Problem- und/oder projektspezifische Erweiterungen
- „Profile“ bieten vorgesehene UML-Erweiterungen an
- Können zur Codegenerierung herangezogen werden
- Basis der „Model Driven Architecture“



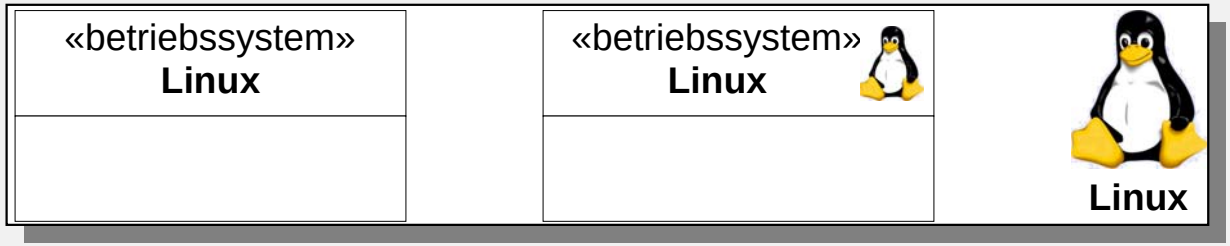
Klassendiagramm



- Stereotypen
  - sind keine Attribute
  - besitzen keinen Wert
  - tragen nicht zum Typ eines Modellelementes bei
  - sind in der Anwendung auf spezielle Modellelemente beschränkt
  - werden nicht direkt in Quellcode abgebildet



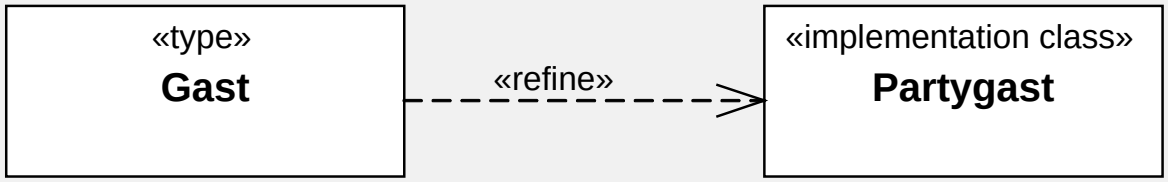
Klassendiagramm

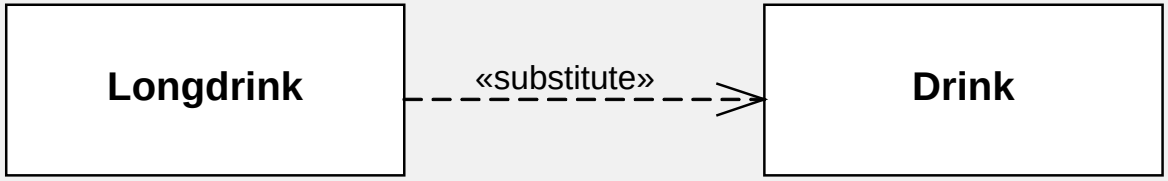


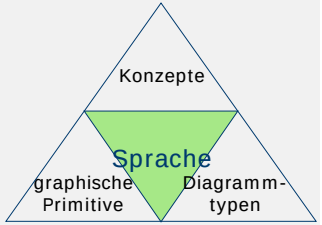
- Achtung: Im obigen Beispiel steht der Pinguin für „Betriebssystem“, nicht für „Linux“!

- Stereotypen
  - können auch graphisch ausgedrückt werden (Die UML-Spezifikation nutzt dies weidlich)
  - können (nach Joos) ...
    - dekorativ
    - restriktiv
    - deskriptiv
    - redefinierend

    ...sein.

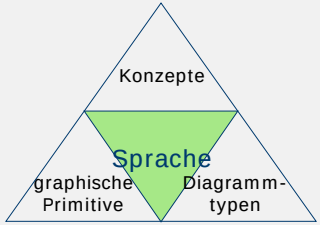




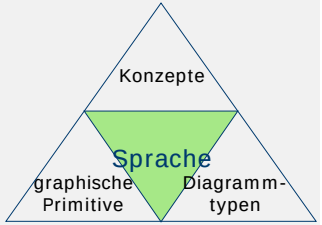


Objektdiagramm

- **Aufgabe**:
  - Darstellung einer (inhärent statischen) Momentaufnahme des Systems zur Laufzeit
- **Aussage**:
  - Zeigt Objekte, Werte und Beziehungsausprägungen
- **Aufgabe im Projekt:**
  - Darstellung von Beispielausprägungen der in den anderen Strukturdiagrammen modellierten Zusammenhänge
- **Änderungen durch UML 2**:
  - Insgesamt: Marginalien!
  - Im Detail ...
    - Offizieller Name in *Instanzdiagramm* geändert
    - Stereotypen copy und become entfernt
    - Multiobjekte existieren nicht mehr

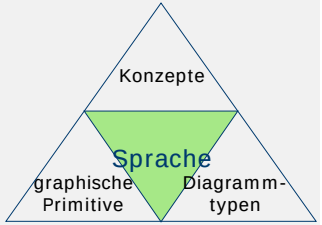


Objektdiagramm

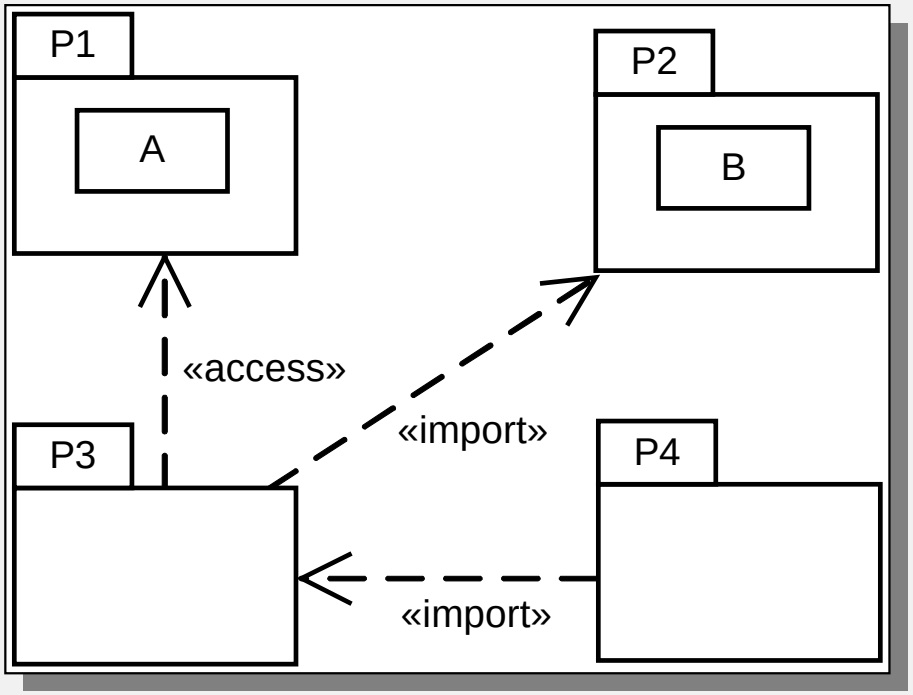


Paketdiagramm

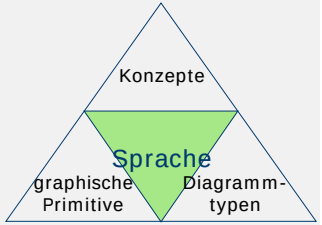
- **Aufgabe**:
  - Darstellung der logischen Organisation von Modellelementen und deren Abhängigkeiten
- **Aussage**:
  - Übersichtliche logische Aufbaustruktur des Systems
- **Aufgabe im Projekt:**
  - Gliederung der Strukturelemente und Dokumentation des Zusammenhangs zwischen den einzelnen Gliederungseinheiten
- **Änderungen durch UML 2**:
  - Insgesamt: keine Änderungen!
  - Jedoch: Der Importmechanismus (insbesondere Behandlung und Verschmelzung von Namensräumen) prominenter dokumentiert.



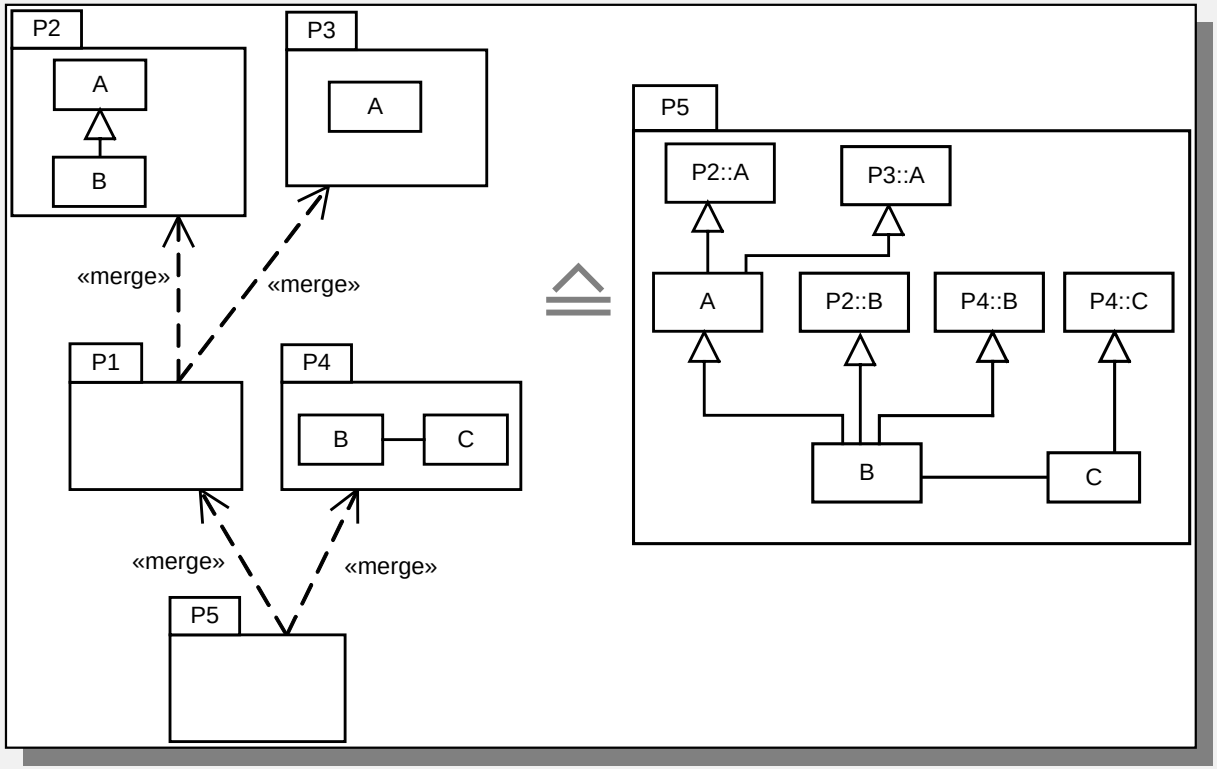
Paketdiagramm



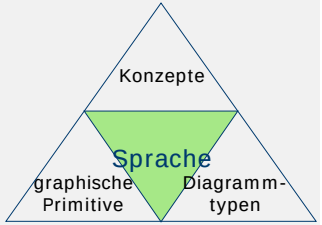
- access:
  - privater Paketimport
    (d.h. importierte Pakete sind im importierenden Paket private)
- import:
  - öffentlicher Paketimport
    (d.h. importierte Pakete sind auch nach außen sichtbar)



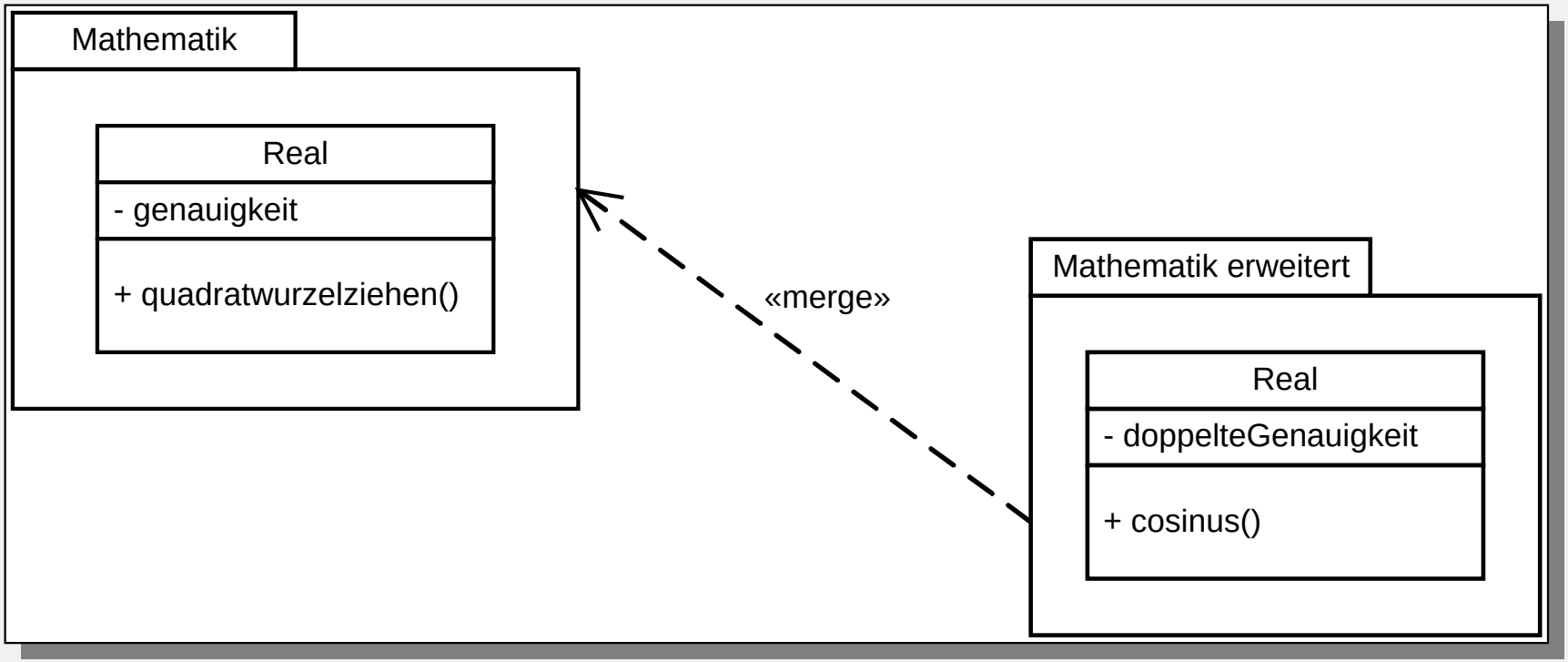
Paketdiagramm

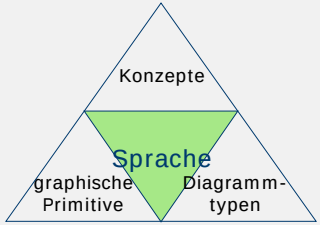


- `access`:
  - privater Paketimport
    (d.h. importierte Pakete sind im importierenden Paket `private`)
- `import`:
  - öffentlicher Paketimport
    (d.h. importierte Pakete sind auch nach außen sichtbar)
- `merge`:
  - Erweiterung von `import`
  - Redefiniert importierte Classifier im aufnehmenden Paket



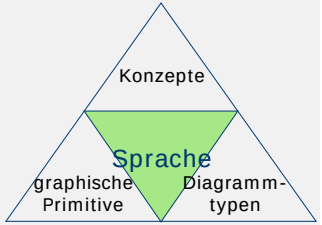
Paketdiagramm





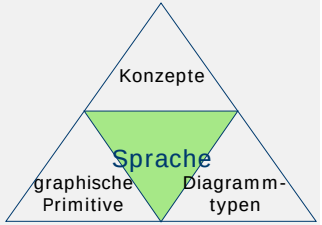
Komponentendiagramm

- **Aufgabe**:
  - Darstellung der Organisation und Abhängigkeiten von Komponenten
- **Aussage**:
  - Darstellung der Struktur und ihrer Erzeugung
- **Aufgabe im Projekt:**
  - Darstellung der physischen Realisierungsstruktur
- **Änderungen durch UML 2**:
  - Komponentendefinition: Sonderform der Klasse
  - Konzept der *Manifestierung*
  - Rechecksnotation entfällt
  - Artefakte können jedes paketierbare Element manifestieren
  - Deploymentspezifikation eingeführt
  - Neue Stereotypen: `device`, `executionEnvironment` und `subsystem`
  - Stereotyp `table` entfernt



Komponentendiagramm

Komplexer Port
Implementierte Schnittstelle
Komponente
«component»
Komponente1
«component»
Komponente2
Bestandteil3
1..*
Bestandteil4
Klasse
Kompositions-konnektor
Bestandteile einer Komponente
Allgemeine Abhängigkeitsbeziehung
Realisierungs-beziehung
«realization»
«artifact»
Komponente6
Bestandteil1
Bestandteil2
«specification»
specDoc
Komponente
Verwendungs-beziehung
«use»
«subsystem»
Komponente3
«component»
Komponente4
Konnektor
«artifact»
Komponente6
«manifest»
Port
Implementierungs-beziehung
Konnektor
Benötigte Schnittstelle
Delegationskonnektor



Komponentendiagramm

«component»
Teilnehmerverwaltung

Sortierter Zugriff

Wahlfreier Zugriff

Paginierer

Satzprogramm

«component»
Listengenerator

«manifest»

«artifact»
Lister_v2.0.ear

Speichermedium

«component»
Datenbankmanagementsystem

«subsystem»
Ein-/Ausgabesystem

«component»
XML-Schnittstelle

Hardwarezugriff

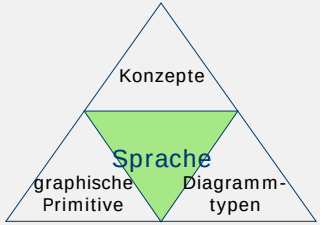
«manifest»

«artifact»
Oracle

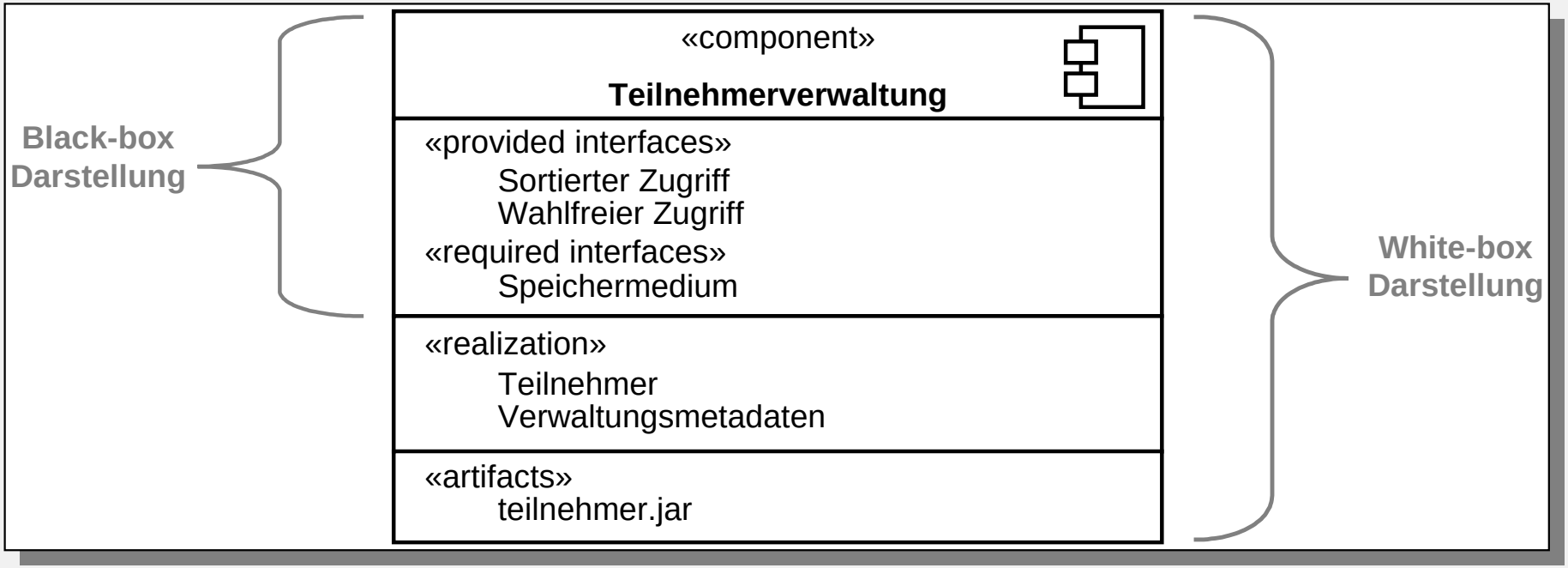
Hardwarezugriff

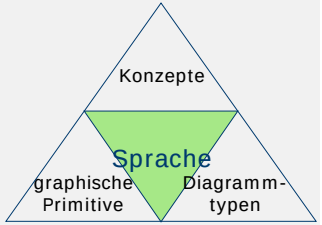
Dateizugriff

«component»
Hardwarezugriffsschicht



Komponentendiagramm





Komponentendiagramm

Zeilennummer

«component»
Zeilengenerator

«interface»
Sortierter Zugriff

«interface»
Speichermedium

«use»

«executable»
util.jar

«manifest»

«component»
Listengenerator

«source»
helper.java

«custom code»

«artifact»
list.class

«source»
list.java

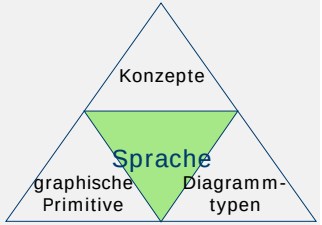
«tool generated»

«component»
Addierer

Zahlenfolge

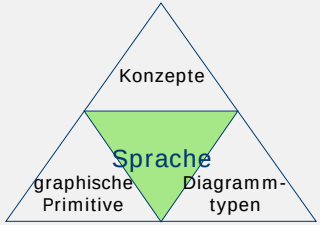
«artifact»
meinKlassendiagramm

«document»
spec.tex

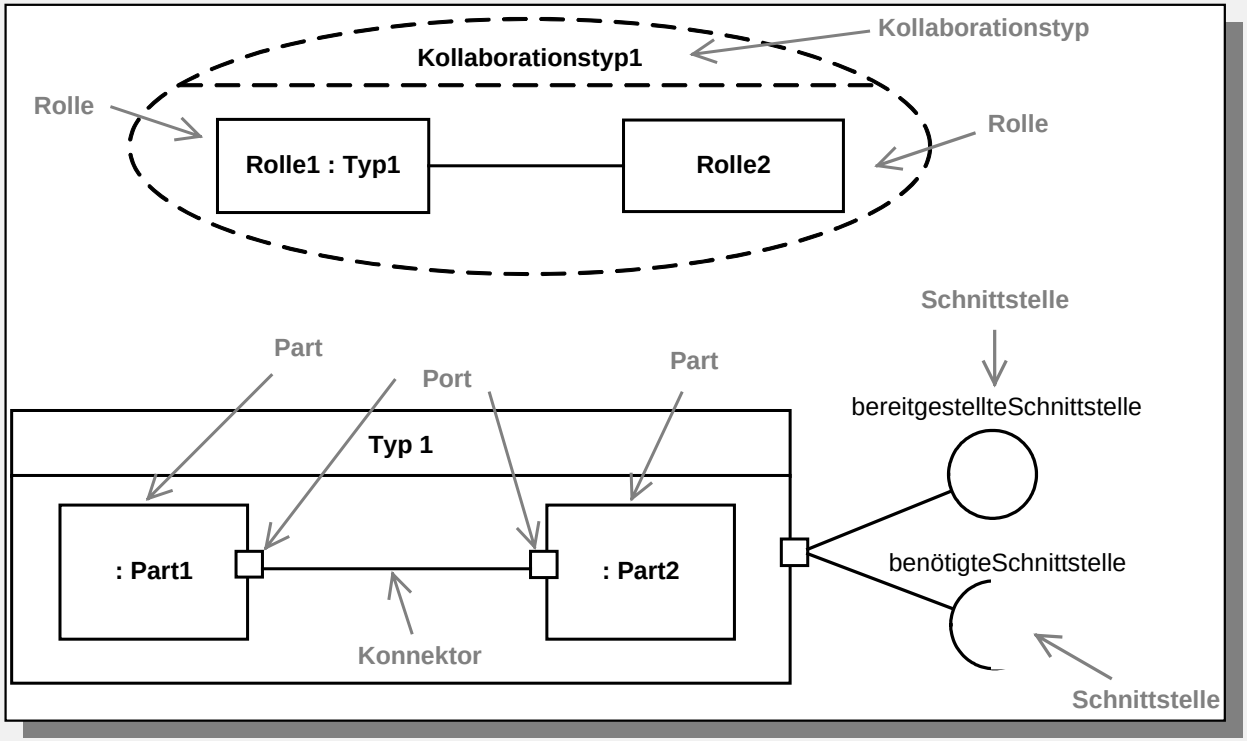


Kompositionsstrukturdiagramm

- **Aufgabe**:
  - Darstellung der internen Struktur eines Classifiers sowie seiner Möglichkeiten zu Interaktion mit anderen Systemkomponenten
- **Aussage**:
  - Struktur und Zusammenspiel der einzelnen Architekturkomponenten
- **Aufgabe im Projekt:**
  - Einsatz in verschiedenen Entwurfsphasen Darstellung der architekturellen Struktur in verschiedenen Detaillierungsebenen
- **Änderungen durch UML 2**:
  - Diagrammtyp neu eingeführt (erweitert das nicht mehr unterstützte Kollaborationsdiagramm)
  - Konnektoren neu eingeführt (aus UML-RT übernommen)
  - Kollaborationstyp und Kollaboration sind eigenständige Modellelemente

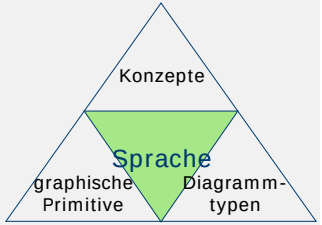


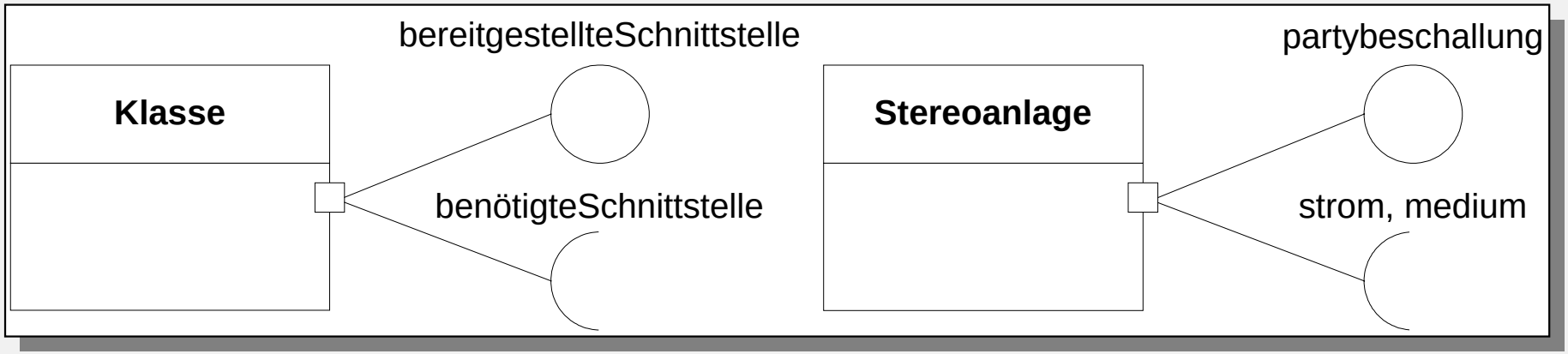
Kompositionsstrukturdiagramm



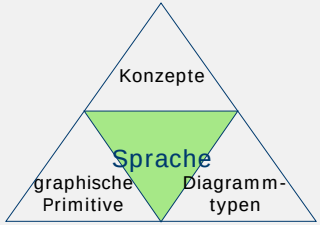
- Part:
  Objekte oder Rollenausprägungen
- Port:
  Öffentlich sicht- und zugreifbarer Interaktionspunkt, der auf einen Operationsaufruf oder den Empfang eines Signals reagiert
- Kollaboration: Zusammenspiel von Operationen oder Classifiern
- Konnektor:
  Assoziationsinstanz (Link), die Kommunikation zwischen (allgemeinen) Instanzen ermöglicht
- Rollenverwendung:
  Bindet realisierende Classifier an Kollaborationsinstanz

Kompositionsstrukturdiagramm

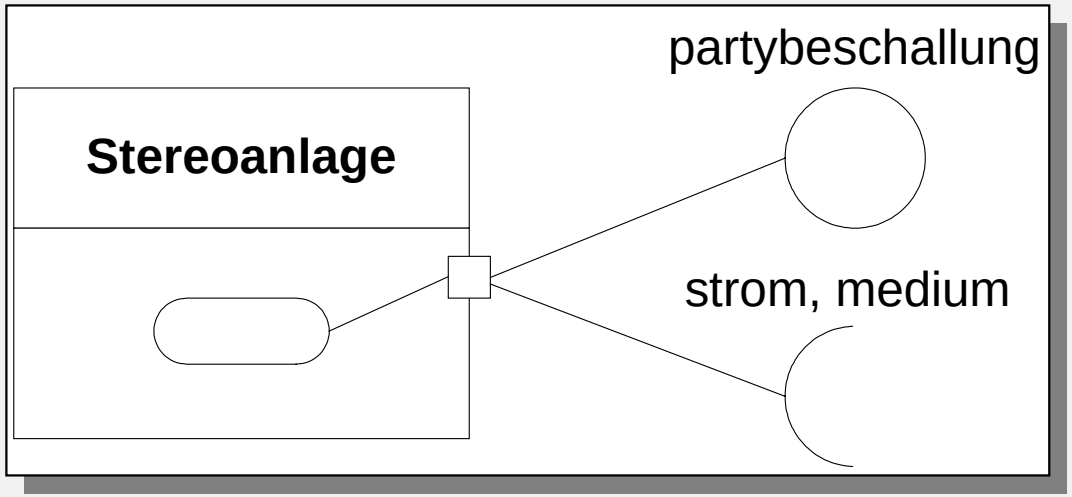




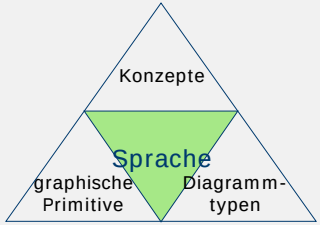
- Port:
  - Definiert einen öffentlich sicht- und zugreifbaren Interaktionspunkt, der auf einen Operationsaufruf oder den Empfang eines Signals reagiert.
  - Werden durch andere Classifier angesprochen.



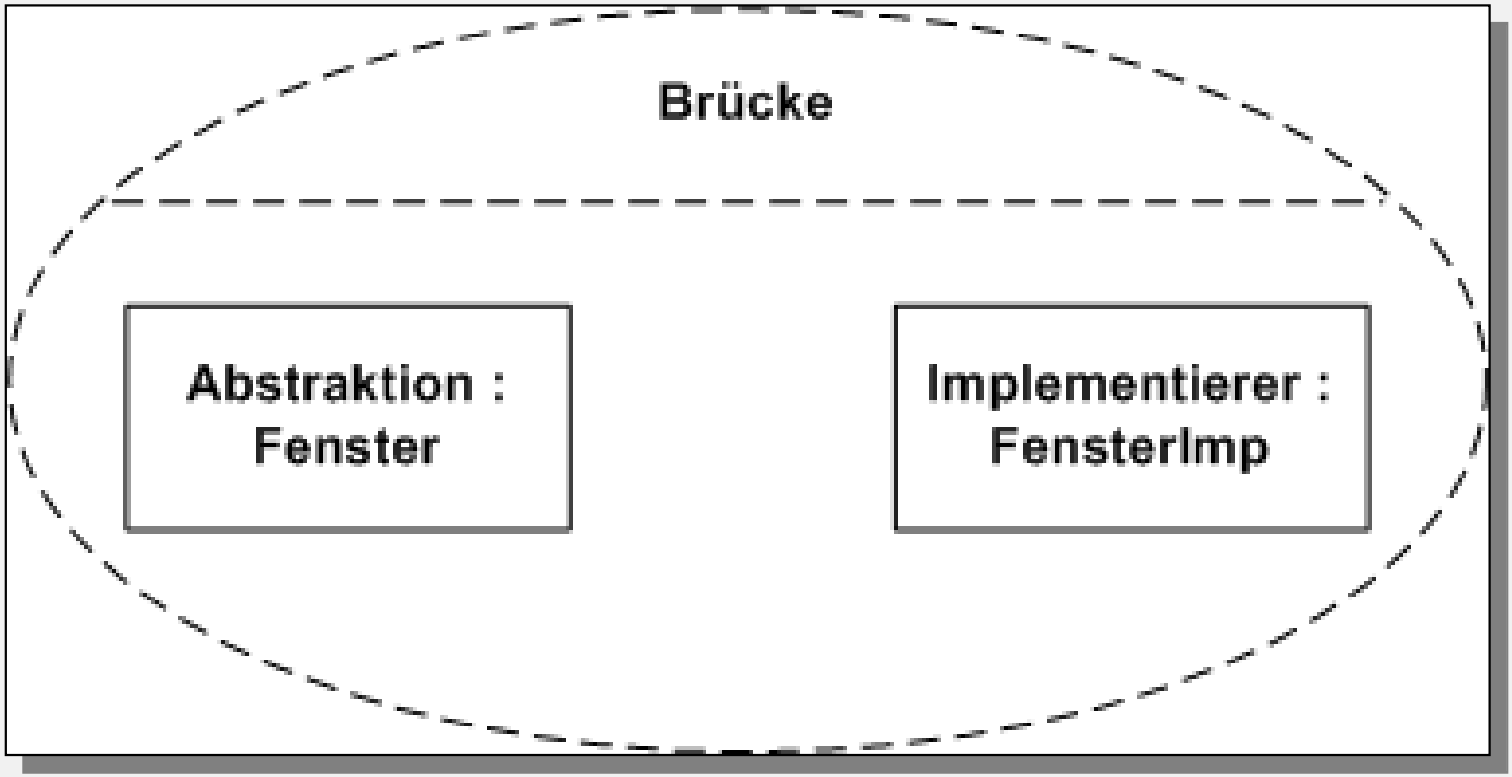
Kompositionsstrukturdiagramm



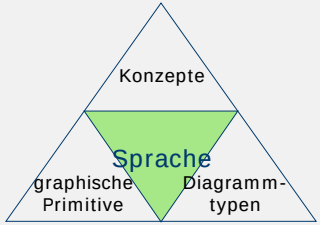
- Behavior Port:
  - Wie *Port*, allerdings mit der Einschränkung, dass der definierende Classifier das Verhalten selbst manifestieren muß, andernfalls ist die empfangende Botschaft oder das empfangene Signal verloren.
  - Kann Sichtbarkeits- und Zugriffsbeschränkt sein.



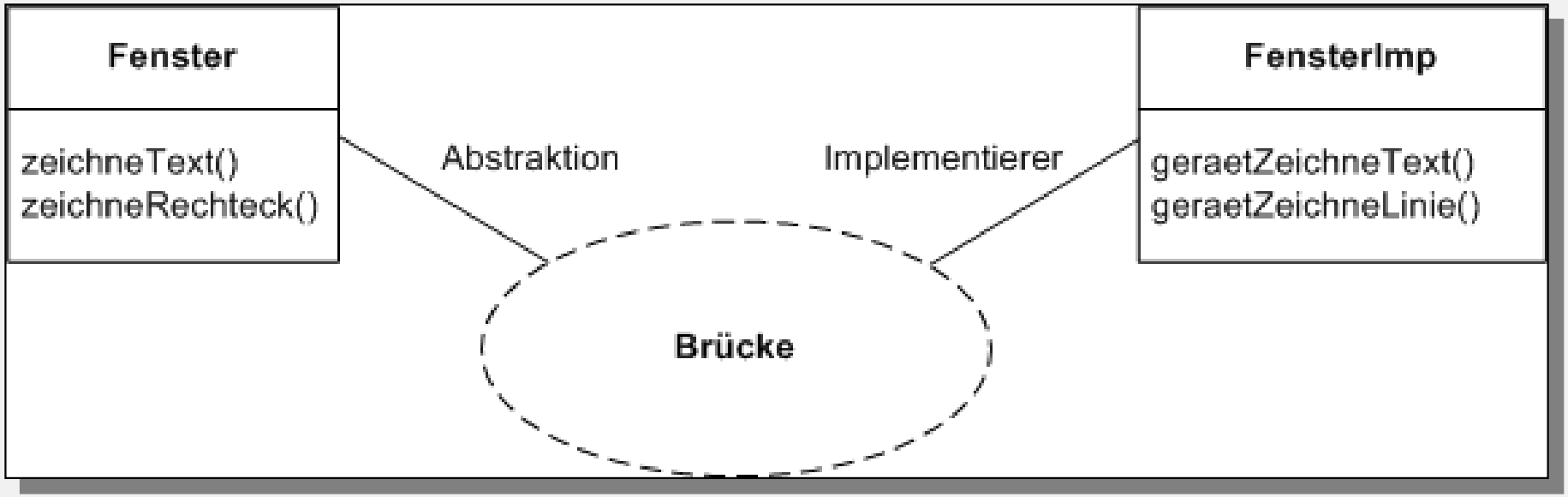
Kompositionsstrukturdiagramm



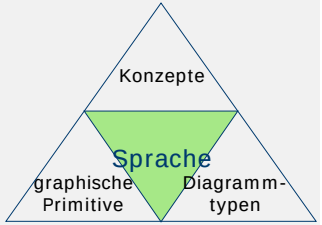
- Kollaboration:
  - Visualisiert das Zusammenspiel von Operationen oder Classifiern



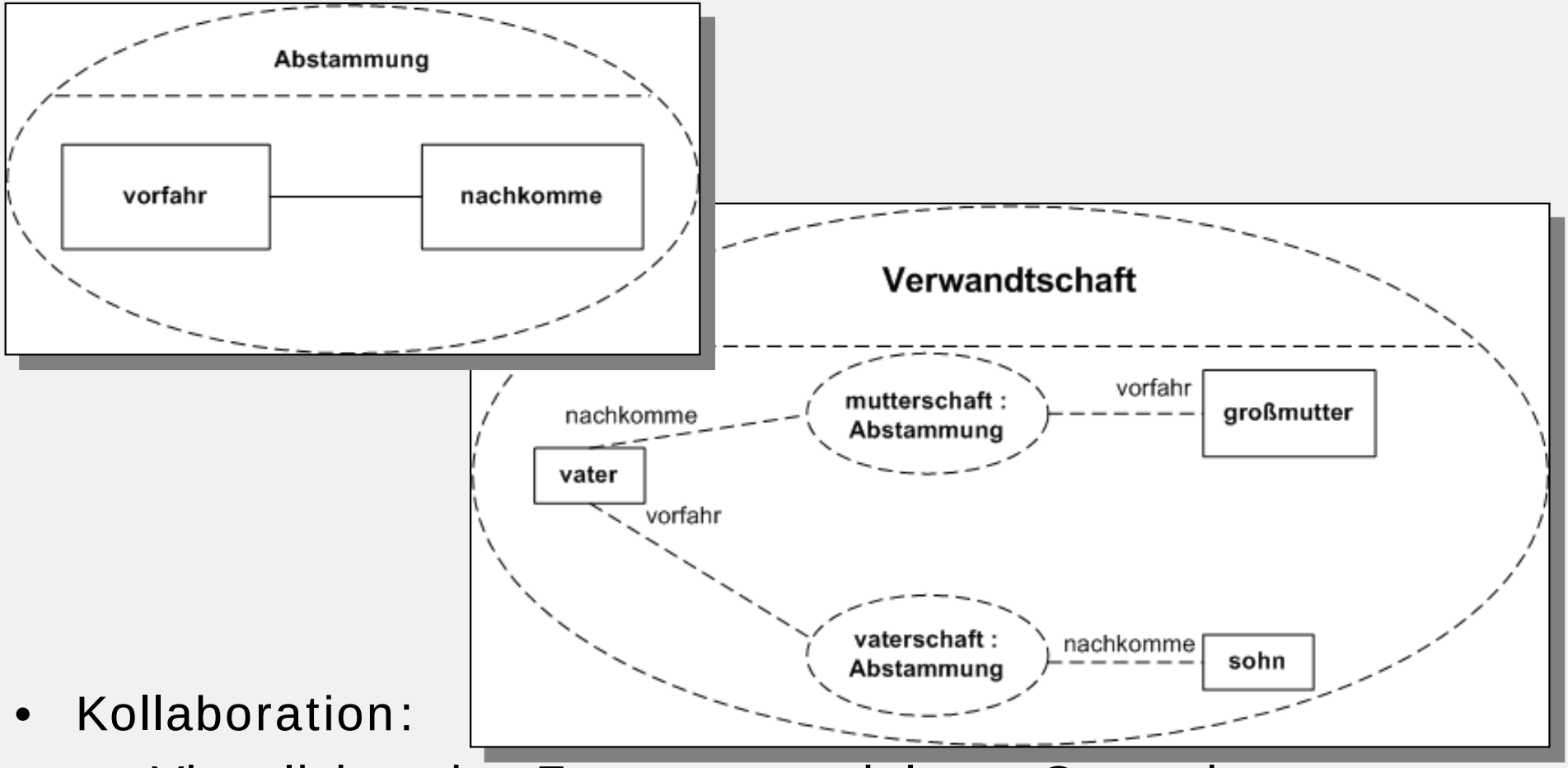
Kompositionsstrukturdiagramm



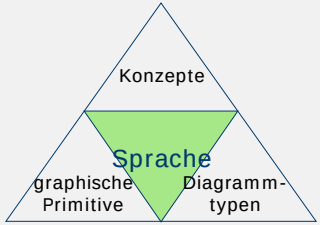
- Kollaboration:
  - Visualisiert das Zusammenspiel von Operationen oder Classifiern



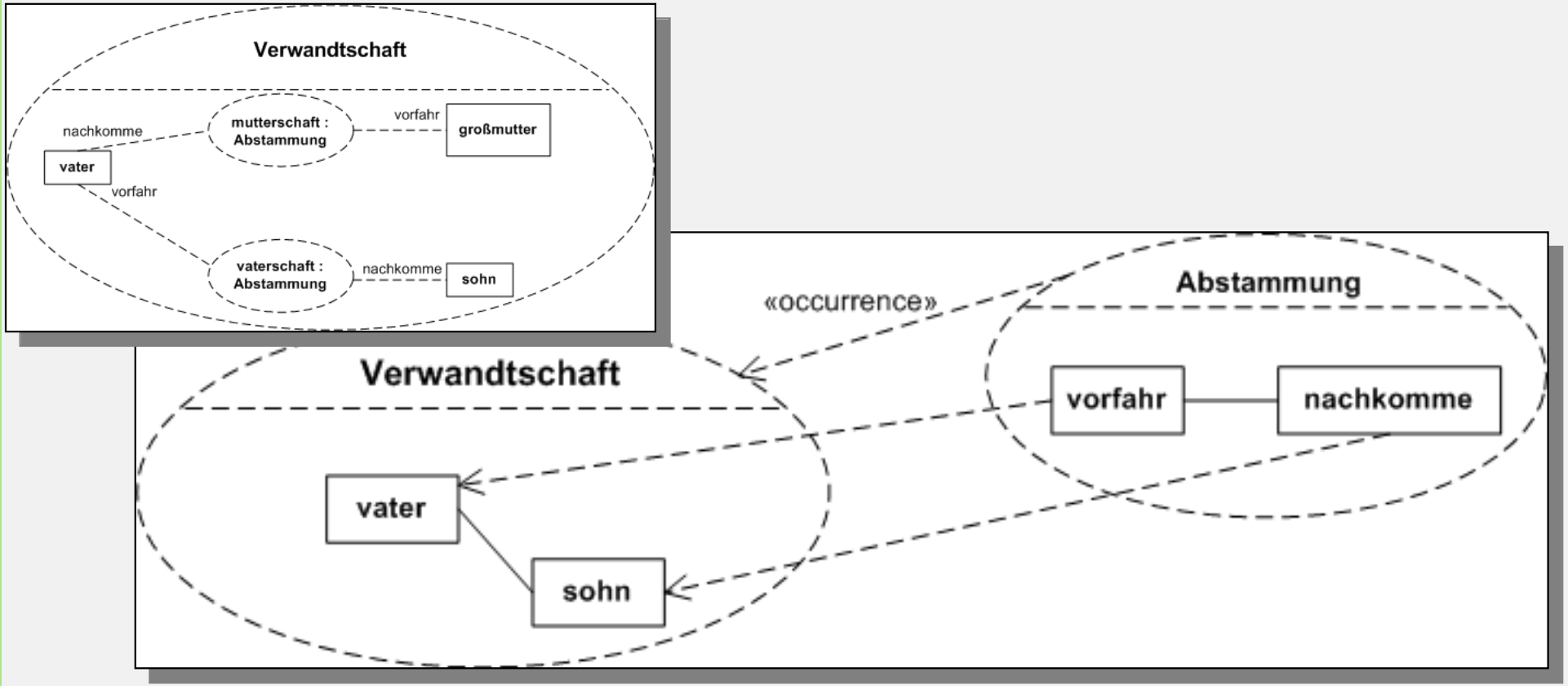
Kompositionsstrukturdiagramm



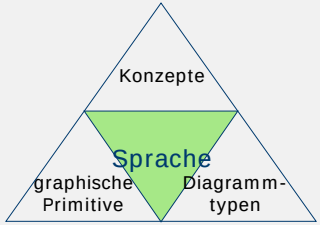
- Kollaboration:
  - Visualisiert das Zusammenspiel von Operationen oder Classifiern



Kompositionsstrukturdiagramm

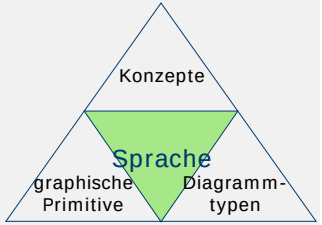


- Kollaboration:
  - Visualisiert das Zusammenspiel von Operationen oder Classifiern

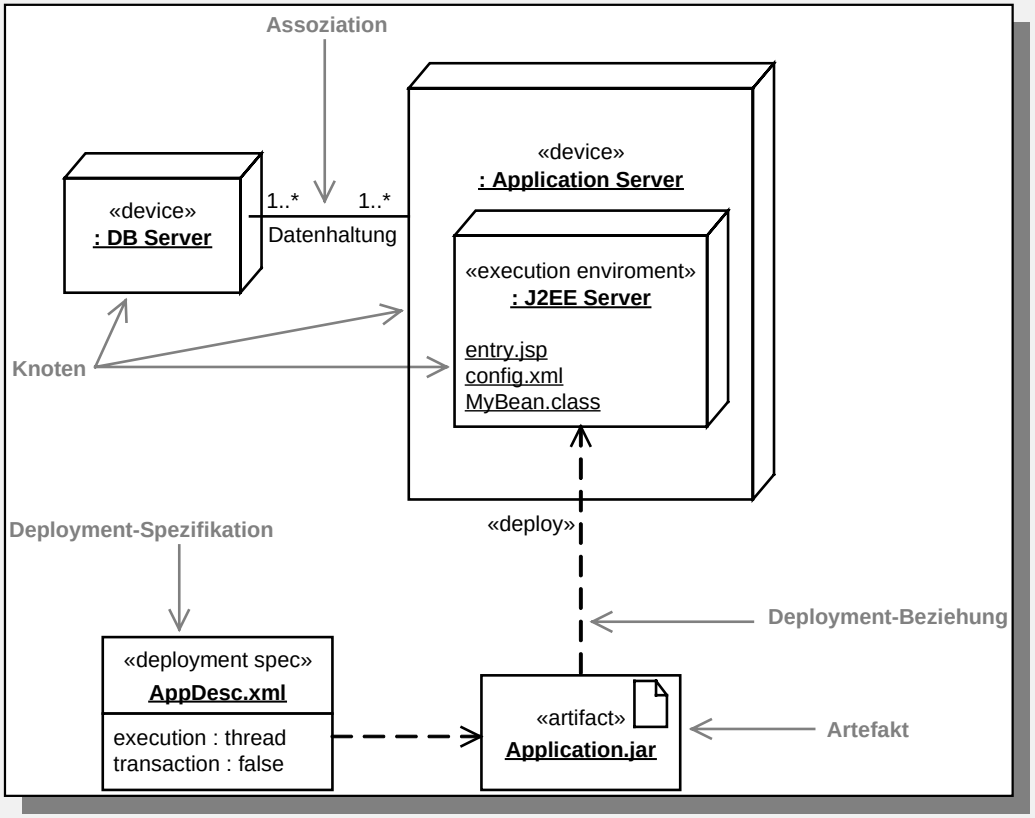


Verteilungsdiagramm

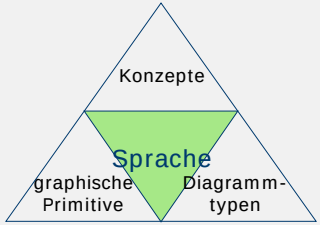
- **Aufgabe**:
  - Zeigt die Ausführungssicht des Systems
- **Aussage**:
  - Art und Lokation der Kompontenverteilung zur Laufzeit
- **Aufgabe im Projekt:**
  - Abstrakte Dokumentation eines verteilten Systems
- **Änderungen durch UML 2**:
  - Neue Primitive: Gerät, Entwicklungsumgebung und Einsatzspezifikation
  - Feinere Spezifikation der Ausführungsknoten
  - Manifestierungs- ersetzt Implementierungsbeziehung
  - Artefakt als Implementierung eines paketierbaren Elements ausdrückbar



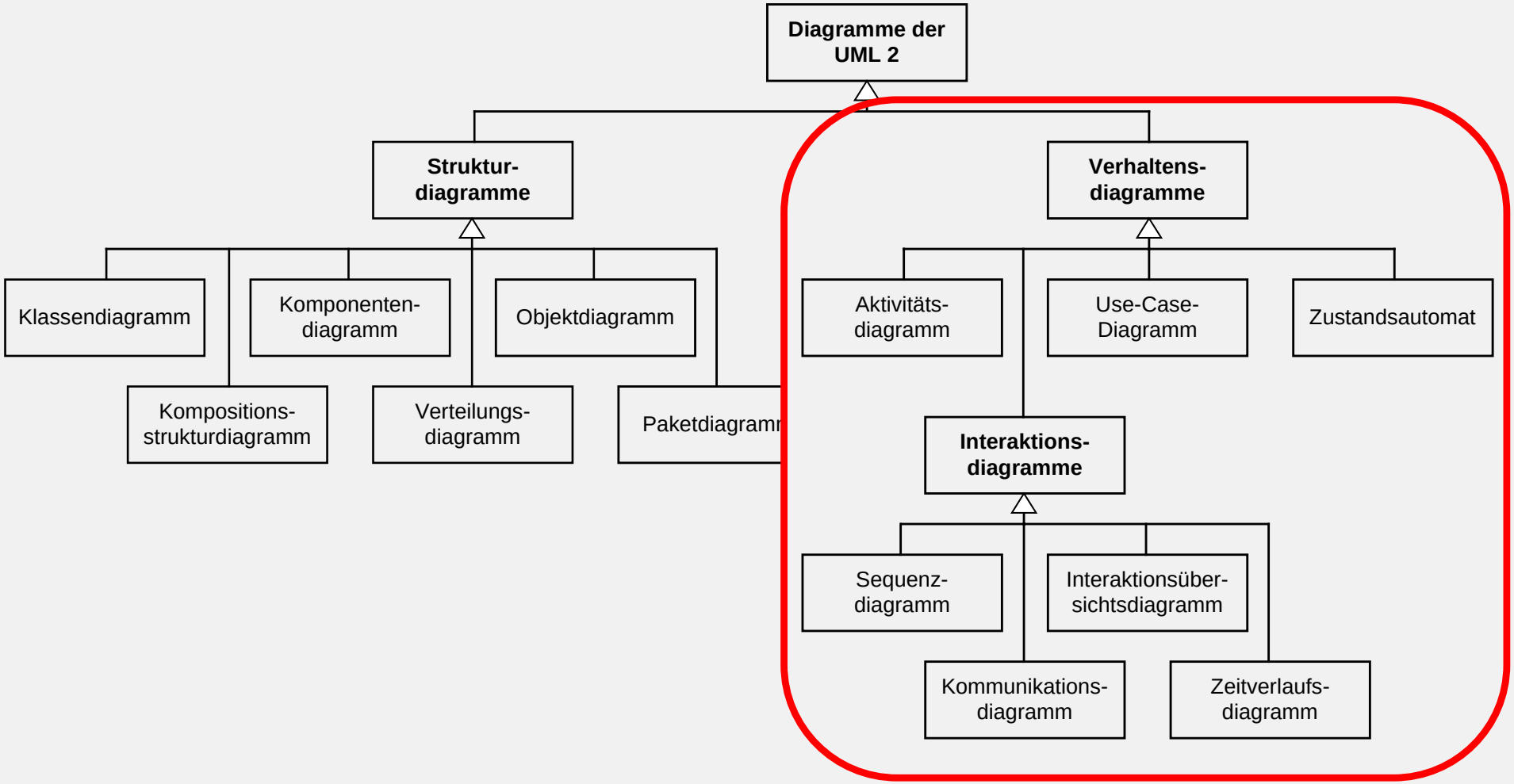
Verteilungsdiagramm

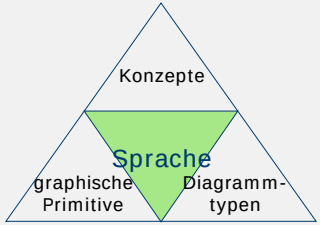


- Basiskonzepte:
  - Artefakt
    Physische Informationseinheit, die während des Entwicklungsprozesses erzeugt oder verwendet wird.
    (z.B. Modelle, Quellcode, Objekdateien, ...)
  - Knoten
    Classifier, der eine zur Ausführungszeit verfügbare Ressource darstellt.
  - Einsatzspezifikation (Deployment Specification)
    Paramtermenge, die Laufzeitverhalten festlegt



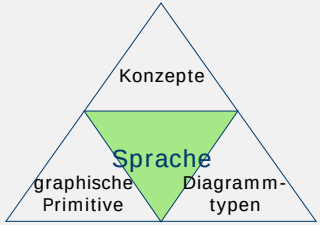
Die Verhaltensdiagramme





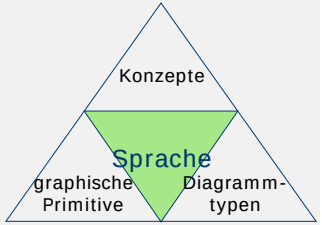
Die Verhaltensdiagramme

- **Use-Case Diagramm**
  Eine abstrakte funktionale Sicht auf das Gesamtsystem aus Sicht des späteren Anwenders
- **Aktivitätsdiagramm**
  Darstellung eines dynamischen Ablaufs
- **Zustandsautomat**
  Beschreibt die internen Zustände eines Classifiers
- **Sequenzdiagramm**
  Beschreibt den Intra- und Intersystem-Datenaustausch
- **Kommunikationsdiagramm**
  Statische Sicht auf dynamische Interaktion
- **Timing-Diagramm**
  Zeitabhängige Zustandsdarstellung
- **Interaktionsübersichtsdiagramm**
  Darstellung des Zusammenspiels verschiedener Interaktionen

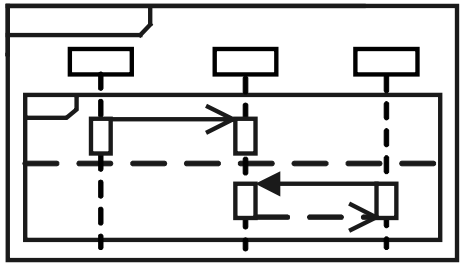


Die Verhaltensdiagramme

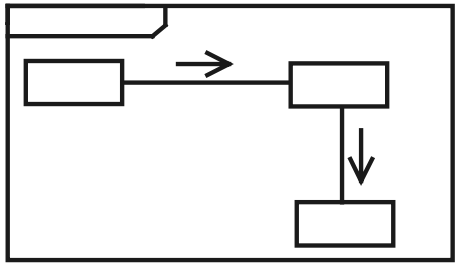
- Hintergrund
  - Dynamische Modellierung seit Ende der 1970er Jahre gängig und verschiedentlich erforscht (z.B. Nassi-Shneiderman, Data Flow Graphs)
  - Die gemeinsame Darstellung statischer und dynamischer Aspekte wird seit dem Aufkommen der ersten OO-Modellierungssprachen (u.a. Boochs OOSE, Rumbaughs OMT) bearbeitet
  - Verhaltensdiagramme historisch jüngster Teil der UML, der ursprünglich sehr losen Bezug zur Strukturmodellierung aufwies
- Jüngere Entwicklungstrends, die eine Neufassung rechtfertigen
  - Echtzeitmodellierung
  - Verbesserte Integration mit statischer Modellierung
  - Wiederverwendung von bestehenden Konstrukten



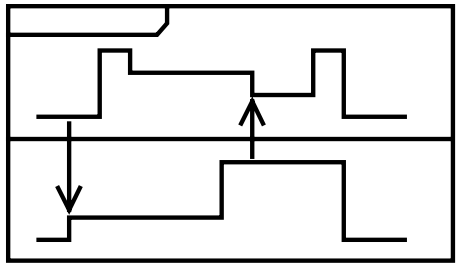
Die Verhaltensdiagramme



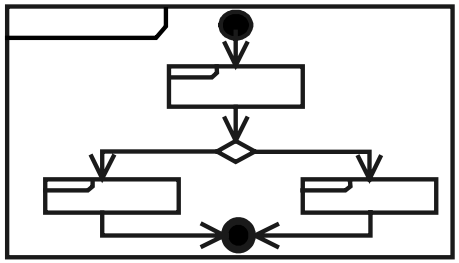
Sequenzdiagramm



Kommunikations-diagramm

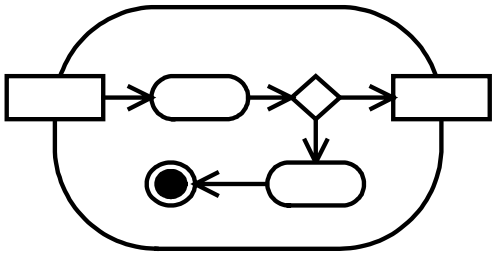


Timing-Diagramm

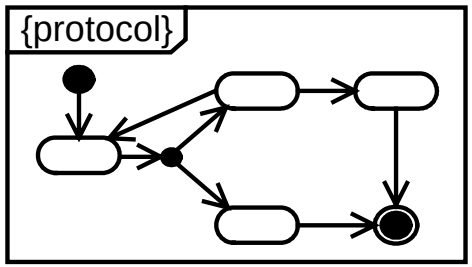


Interaktionsübersichts-diagramm

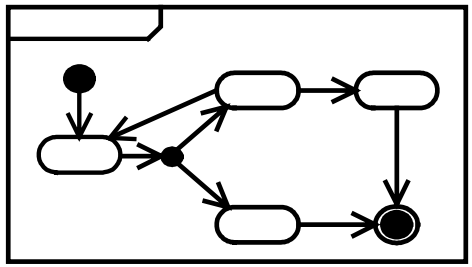
Interaktions-diagramme



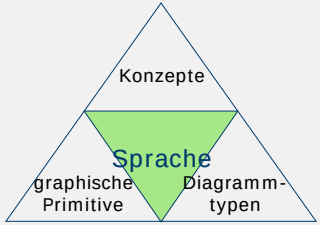
Aktivität



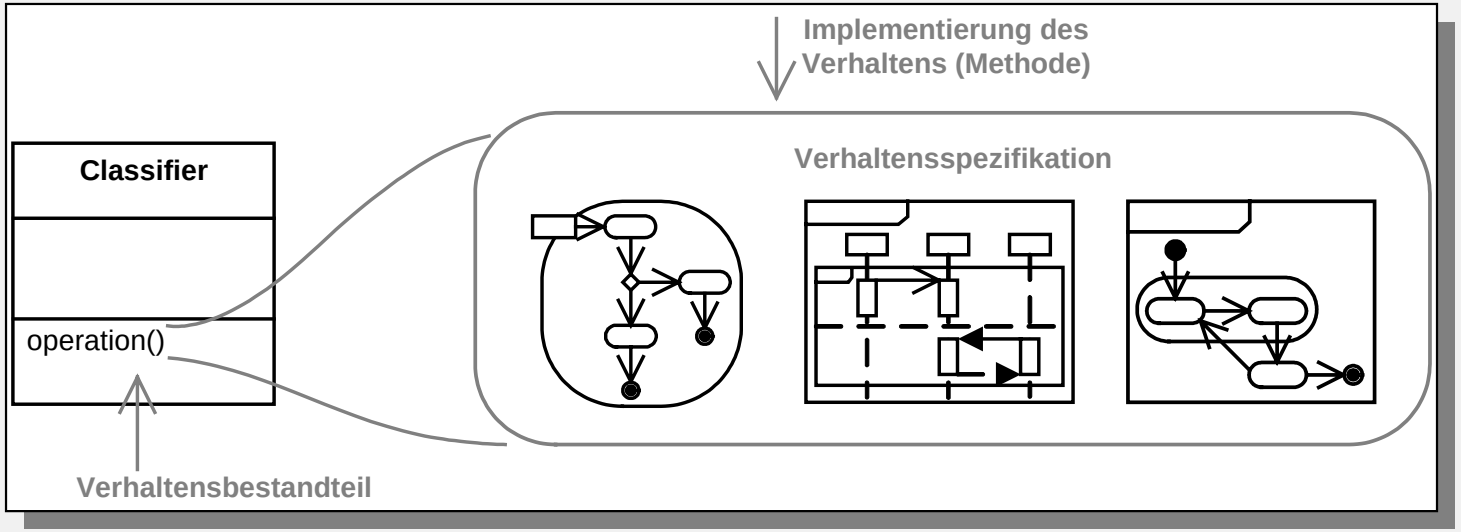
Protokoll-zustandsautomat



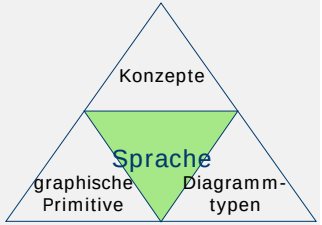
Verhaltens-zustandsautomat



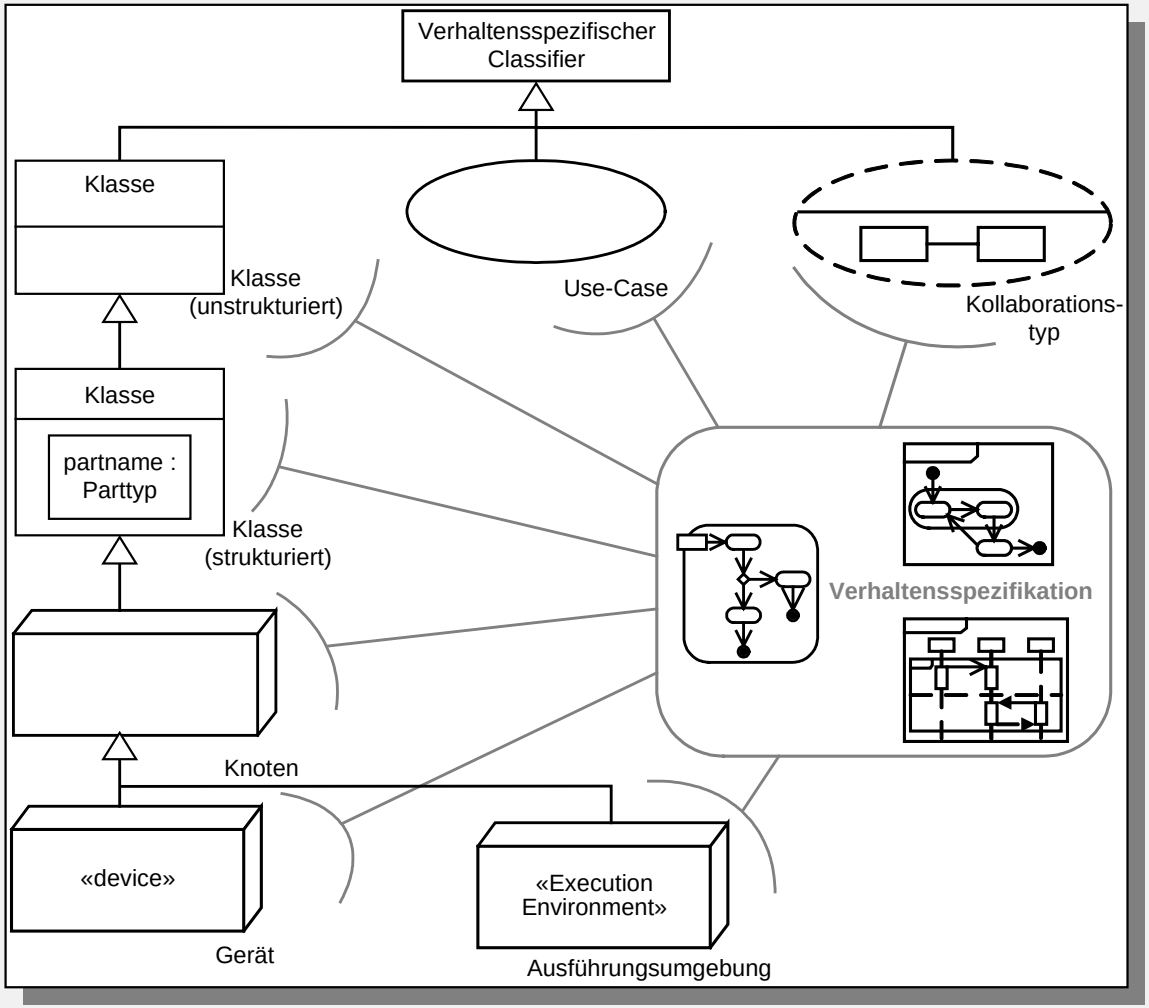
Die Verhaltensdiagramme



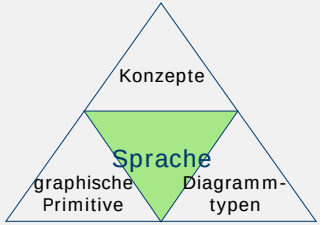
- Das Verhalten jeder Operation kann durch ein Diagramm dynamischen Typs spezifiziert werden



Die Verhaltensdiagramme

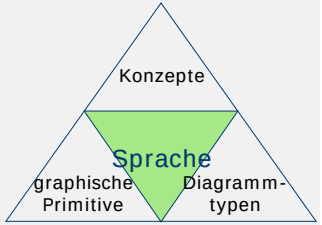


- Die einer Verhaltensspezifikation zugrundeliegenden statischen Zusammenhänge können aus allen Strukturdiagrammen entnommen sein

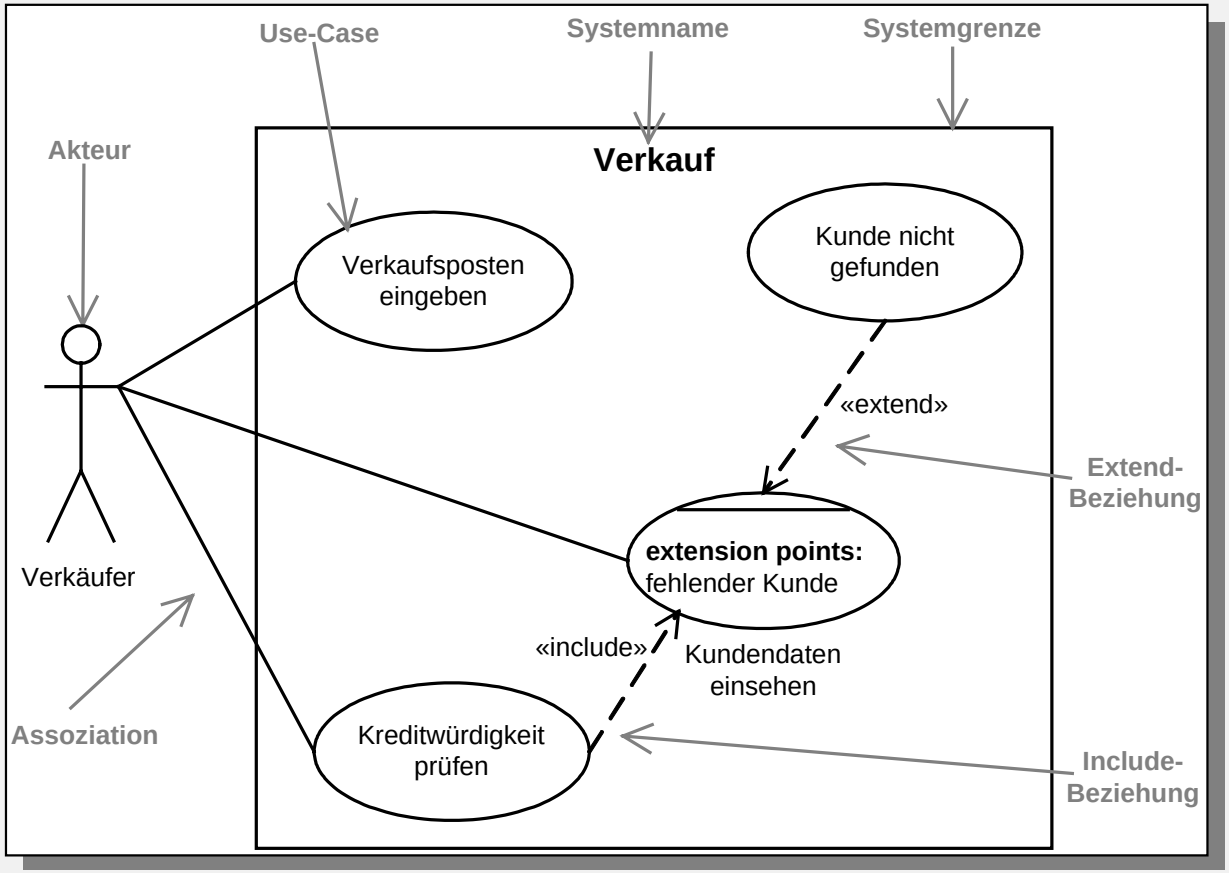


Use-Case-Diagramm

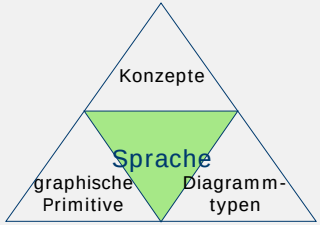
- **Aufgabe**:
  - Darstellung einer abstrakten funktionalen Sicht auf das Gesamtsystem aus Sicht des späteren Anwenders
- **Aussage**:
  - Technikfern spezifizierter Leistungsumfang des Systems
- **Aufgabe im Projekt:**
  - Dokumentation erster früher Analyseergebnisse
- **Änderungen durch UML 2**:
  - Akteur muß zwingend benannt sein
  - Vorbedingungen und Erweiterungspunkte werden als Notiz notiert
  - Classifier können Use Cases besitzen
  - Classifier können Use Cases realisieren



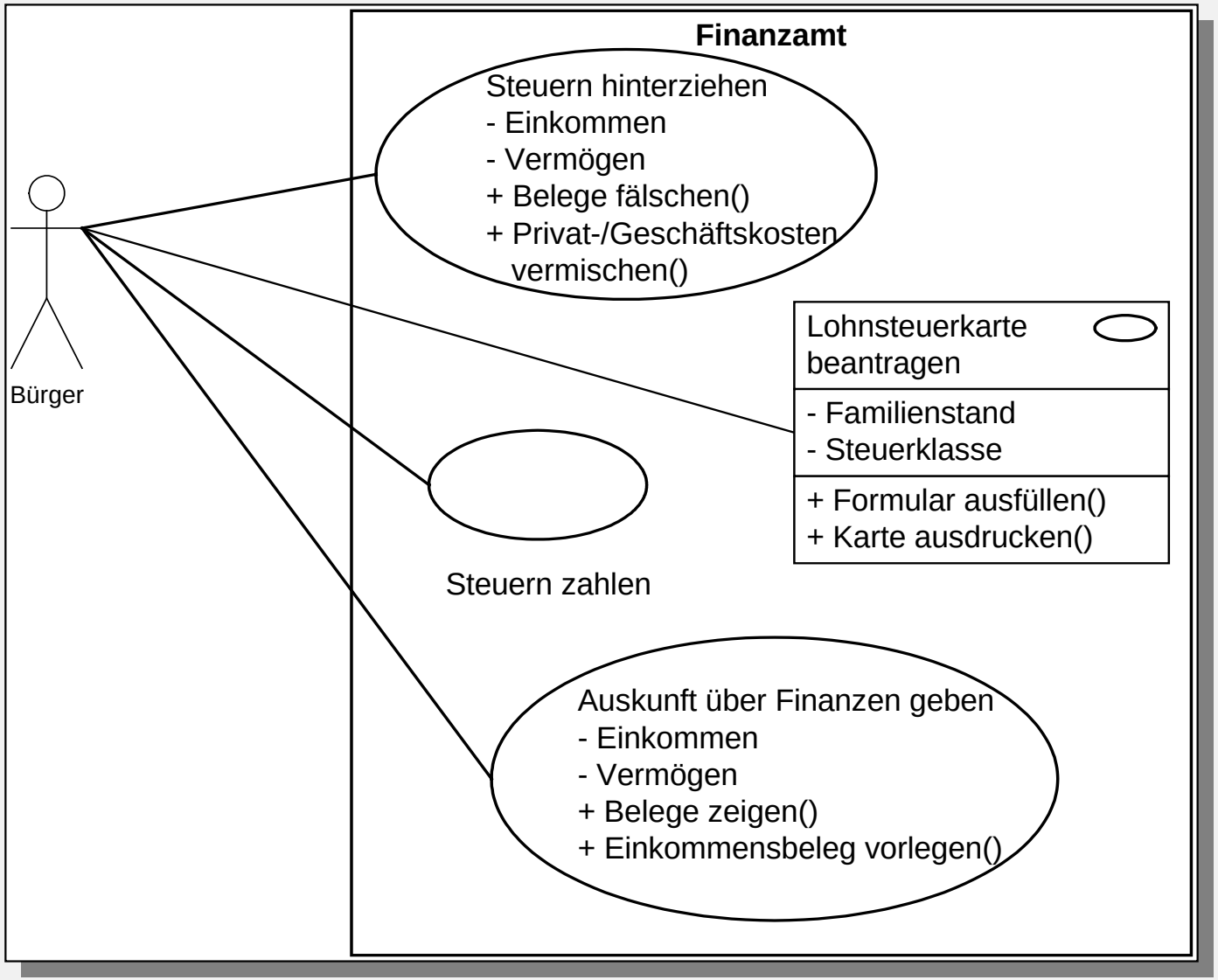
Use-Case-Diagramm

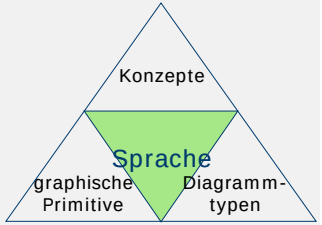


- System (Betrachtungsgegenstand):
  - Umgrenzt die Einheit, welche die Use-Cases realisiert
  - Darstellung ist nicht zwingend notwendig
- Assoziationen:
  - beschreiben die Beziehungen zwischen Akteuren und Use-Cases
  - `extend`-Beziehung: Verhalten eines Use-Case kann durch einen anderen erweitert werden
  - `include`-Beziehung: Verhalten eines Use-Case ist vollständig in einem anderen enthalten



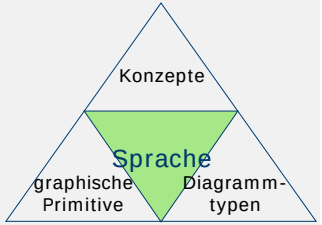
Use-Case-Diagramm



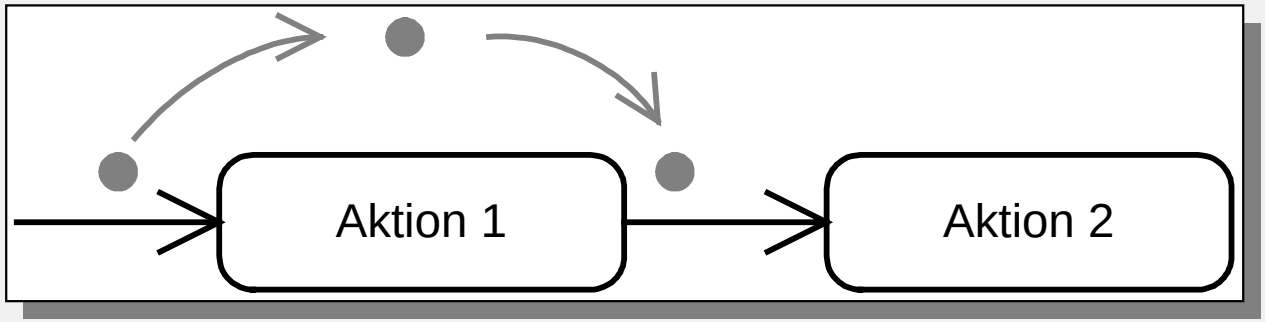


Aktivitätsdiagramm

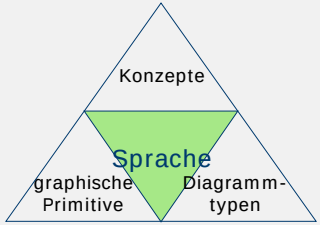
- **Aufgabe**:
  - Darstellung eines dynamischen Ablaufs
- **Aussage**:
  - Realisierung eines bestimmten Verhaltens durch das System
- **Aufgabe im Projekt:**
  - Geschäftsprozeßmodellierung
  - Beschreibung von Use Cases
  - Dokumentation der Implementierung einer Operation
- **Änderungen durch UML 2**:
  - Aktivitäten unabhängig von Zustandsautomaten
  - Petri-Netz-ähnliche Semantik
  - Diagrammtyp wird als *Aktivität* bezeichnet
  - Vor- und Nachbedingungen
  - Notation der Aktion und des Zustandes vereinheitlicht
  - Multiple Startknoten
  - Verschiedene End(-knoten)-Semantiken
  - Neue Notationselemente
  - ...



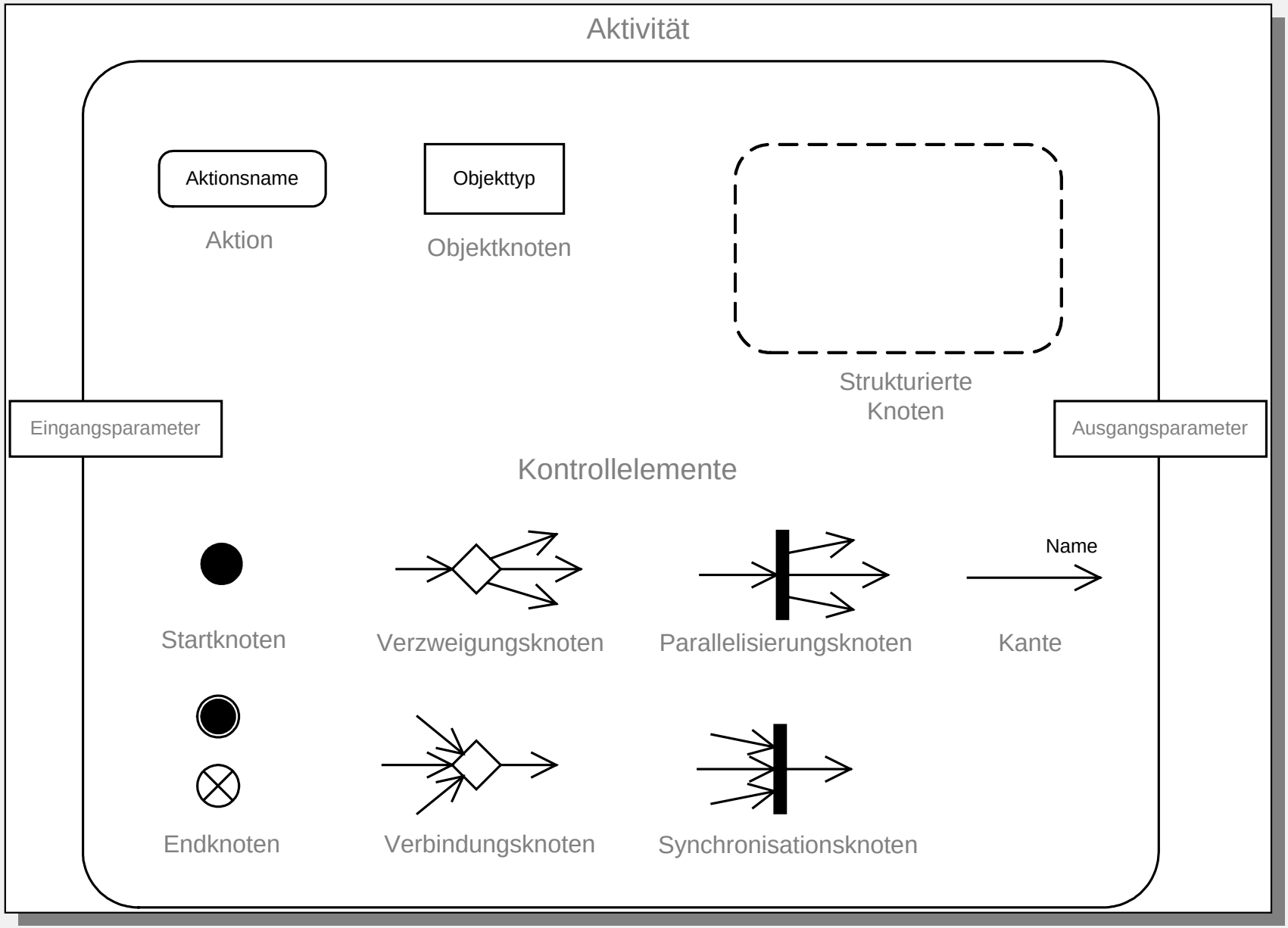
Aktivitätsdiagramm

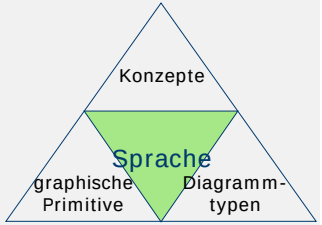


- Tokenkonzept
  - aus den Petri-Netzen übernommen
  - Tokenfluss steuert Ablauf einer Aktivität
  - Ermöglicht die präzise Beschreibung des Verhaltens
  - nur gedankliches Konstrukt (keine explizite Modellierung)

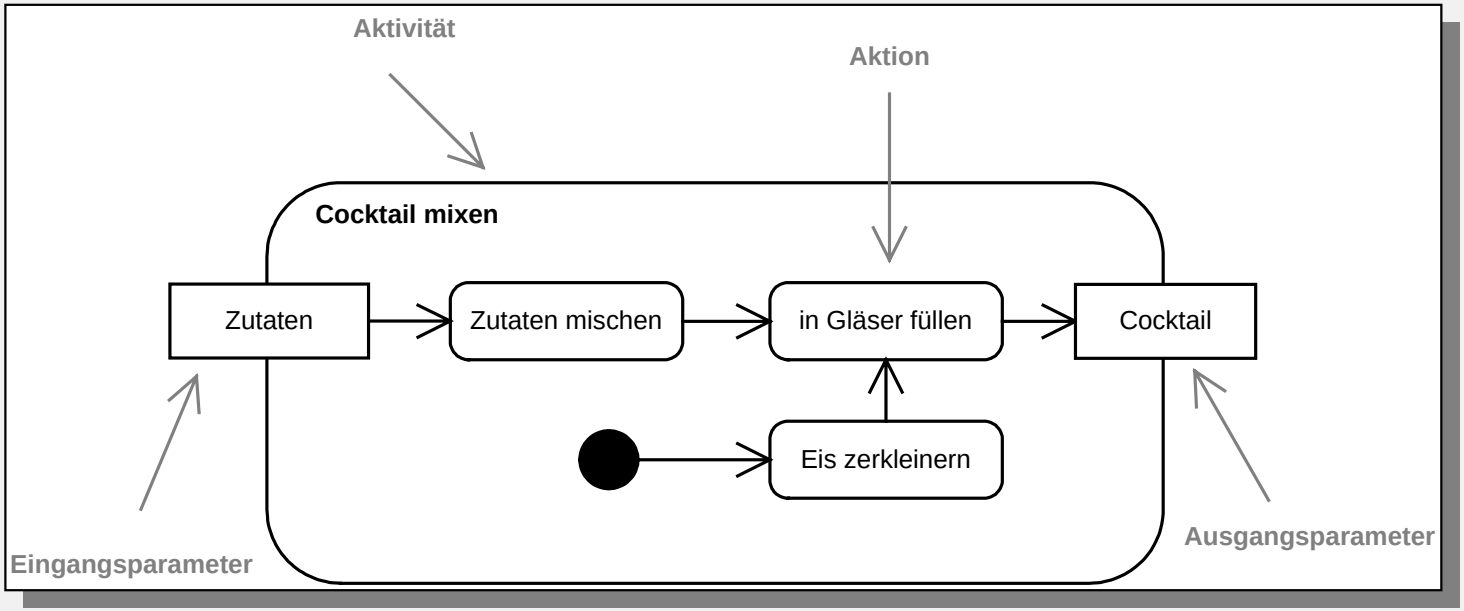


Aktivitätsdiagramm

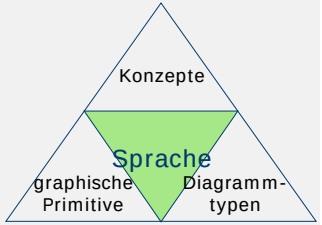




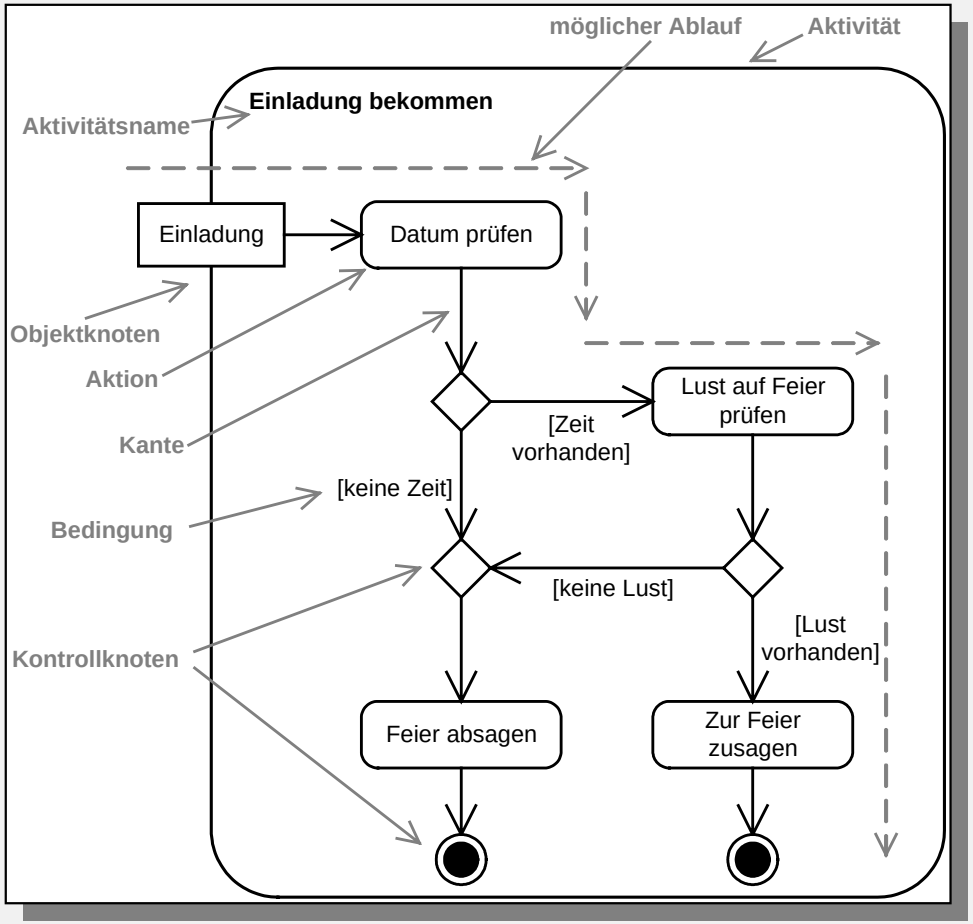
Aktivitätsdiagramm



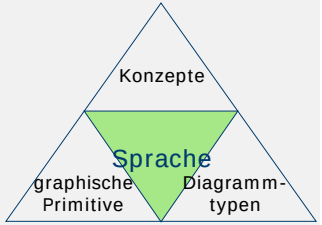
- Diagrammtyp heißt *Aktivität*
- Eine Aktivität kann Ein- und Ausgangsparameter besitzen
- Aktionen sind Verhaltensaufrufe
- Summe der Aktionen realisiert die Aktivität



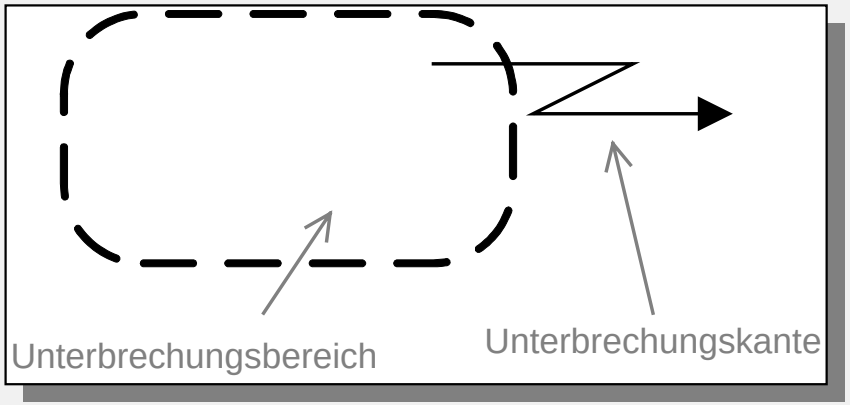
Aktivitätsdiagramm

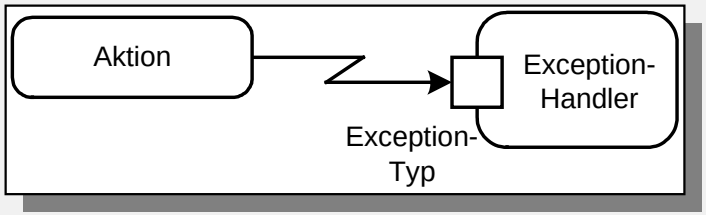


- Kontrollelemente
  - steuern den Ablauf der Aktivität
  - starten und beenden Abläufe
  - ermöglichen Nebenläufigkeit
  - dienen der Synchronisation
  - lassen alternative Abläufe zu

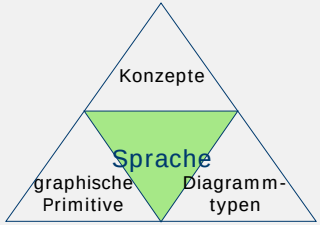


Aktivitätsdiagramm

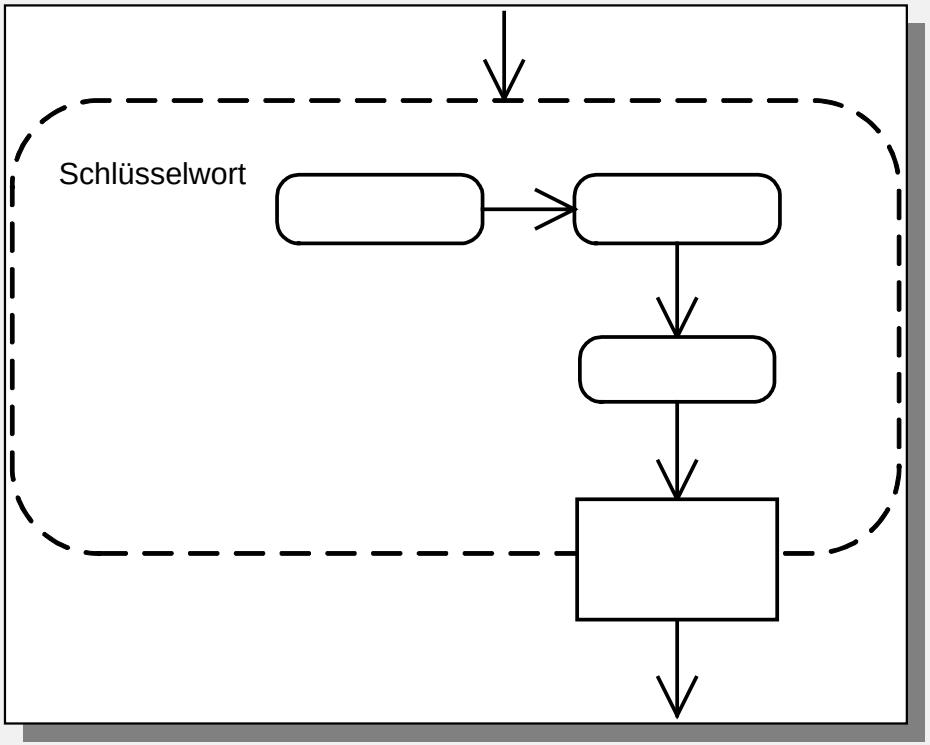




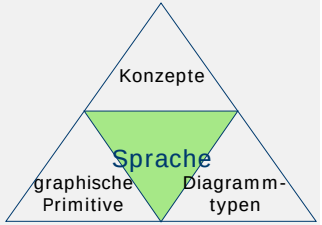
- Unterbrechungsbereich:
  - Beinhaltet eine Menge von Aktionen
  - Kann über Unterbrechungskante verlassen werden. Alle Aktionen im Bereich werden dann beendet.
- Exception-Handler:
  - Ermöglicht die Beschreibung von Ausnahmen
  - Exception-Handler substituiert eine Aktion



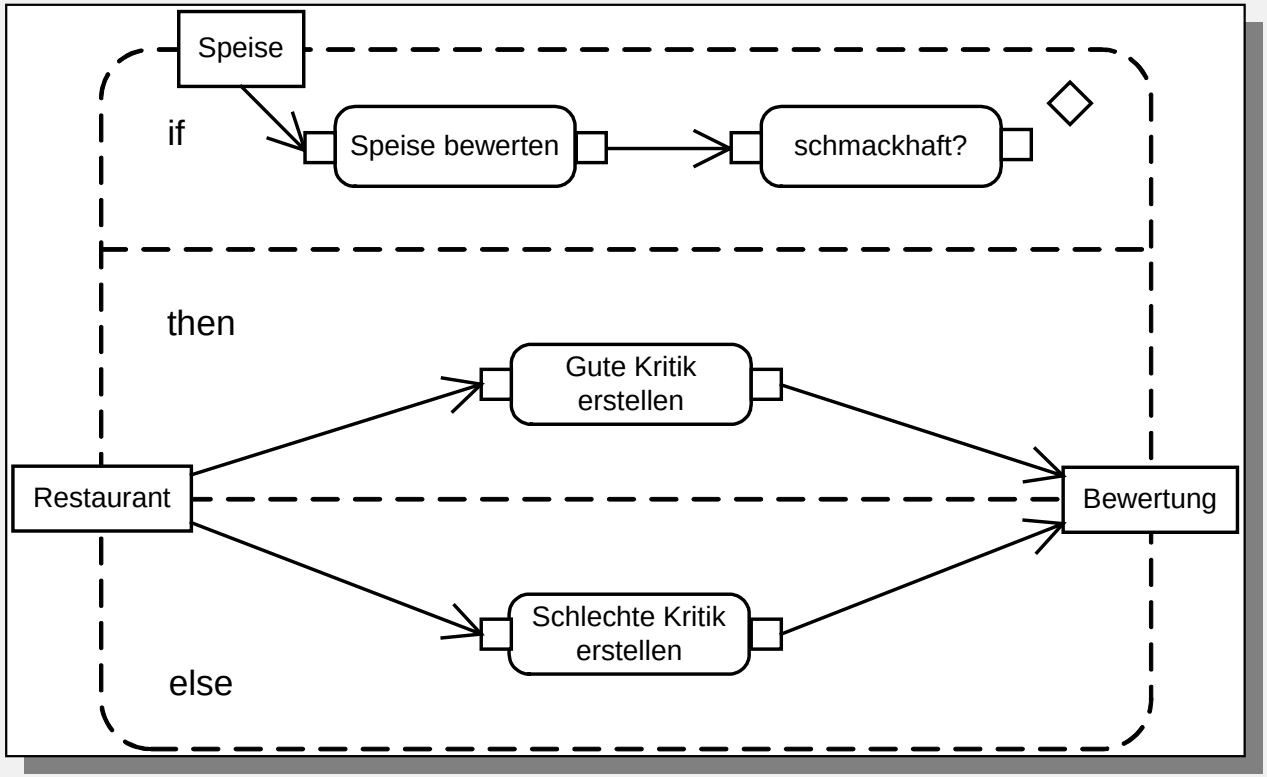
Aktivitätsdiagramm



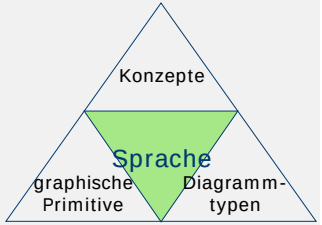
- Strukturierte Knoten:
  - Umfassen Ausschnitt einer Aktivität
  - Ausführung startet mit dem anliegen aller Token der Eingangsknoten
  - Objektknoten als Ein- und Ausgangsparameter möglich



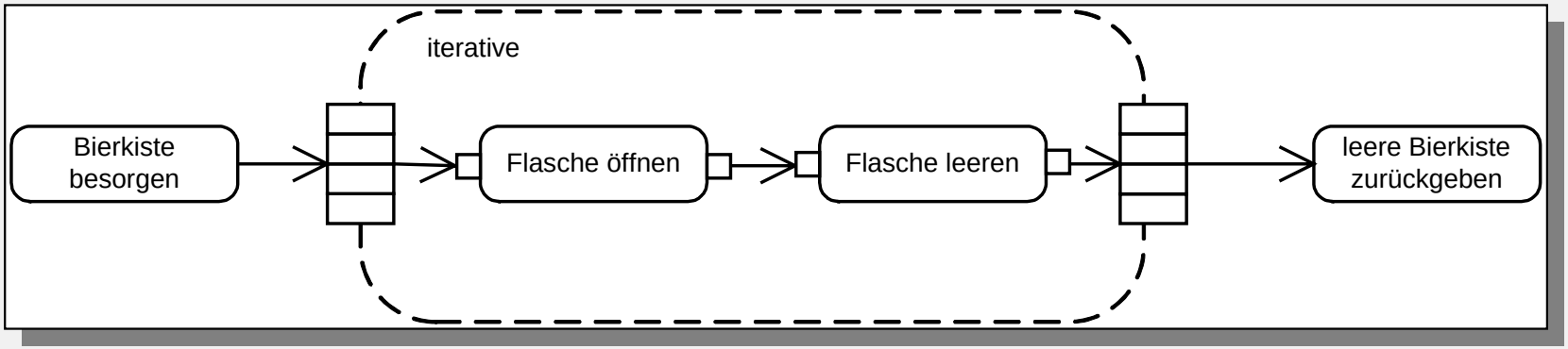
Aktivitätsdiagramm



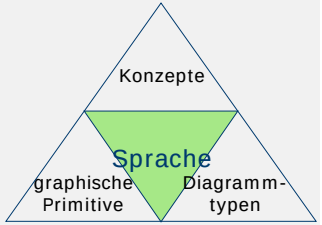
- Strukturierte Knoten zur Visualisierung komplexer Entscheidungen
- `if`: prüfen der Bedingung
- `then`: auszuführende Elemente
- `else`: möglicher Ablauf, wenn kein if-Bereich zutrifft
- `else if`: wie `if`-Bereich nur mit vorgegebener Prüfreihenfolge



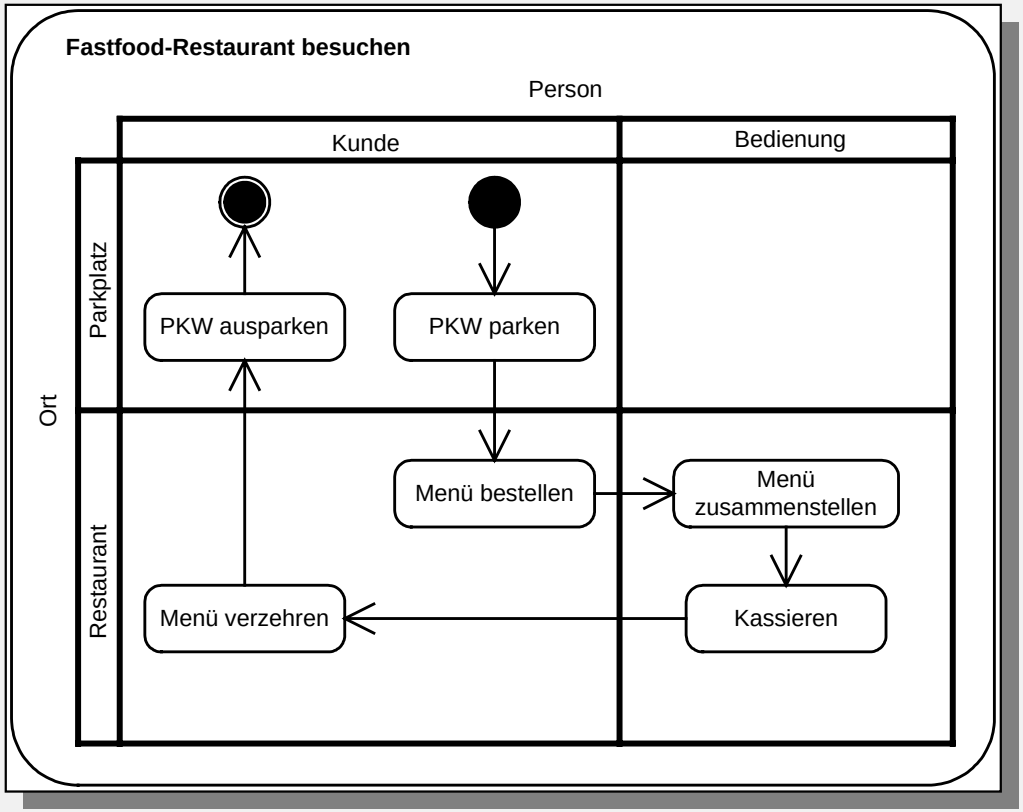
Aktivitätsdiagramm



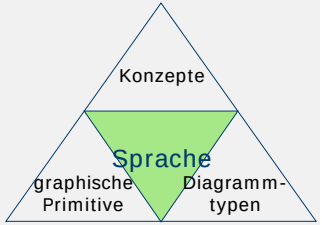
- Mengenverarbeitung
  - Einzelne Betrachtung der Elemente welche in der restlichen Aktivität nur als Sammlung betrachtet werden
  - z.B. Listen, Vektoren, `hashtable`...
  - Elemente werden als Objektknoten (Pin) übergeben



Aktivitätsdiagramm

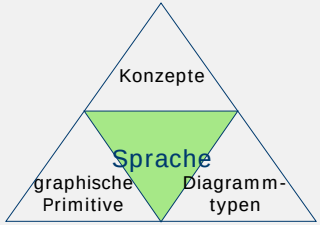


- Aktivitätsbereiche
  - Teilung des Diagramms logisch gruppierte Partitionen
  - Hierarchische und mehrdimensionale Partitionierung möglich

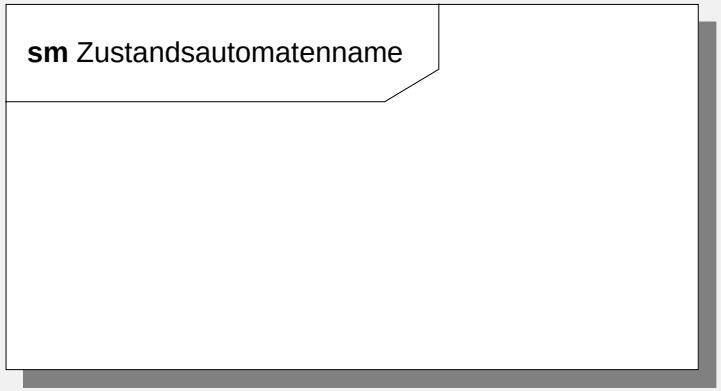


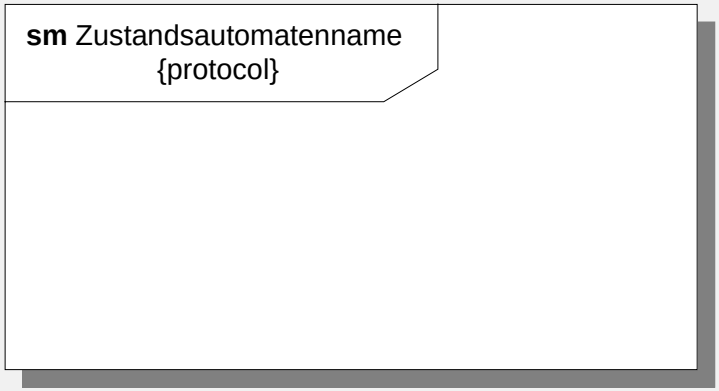
Zustandsdiagramm

- **Aufgabe**:
  - Beschreibt die internen Zustände eines Classifiers
- **Aussage**:
  - Zugelassene Status eines Classifiers durch Betrachtung als Zustandsautomat
- **Aufgabe im Projekt:**
  - Zustandbeschreibung eines Classifiers
  - Detaillierung eines Use Cases
  - Verhaltensbeschreibung einer extern angebotenen Schnittstelle
- **Änderungen durch UML 2**:
  - *Protokollzustandsautomat* neu eingeführt (Spezialisierung des allgemeinen Zustandsdiagramms)
  - Explizierung von Ein- und Austrittspunkten sowie Terminatoren
  - Vererbungssemantik (Overriding und Extension) geregelt

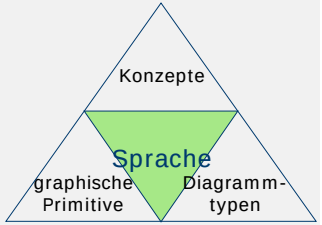


Zustandsdiagramm





- Unterscheidung:
  - Verhaltenszustandsautomaten (Zustandsdiagramm)
  - Protokollzustandsautomaten
- Ein Verhaltenszustandsautomat bildet das diskrete Verhalten einer Instanz eines Classifiers ab.
- Ein Protokollzustandsautomat beschreibt die erlaubte Aufrufsabfolge der Instanz eines Classifiers.



Zustandsdiagramm

sm Zustandsautomat

Zustandsname

einfacher Zustand

Zustandsname

zusammengesetzter Zustand

Unterzustandsautomatenzustand : sm Unterzustandsautomat

Unterzustandsautomatenzustand

Trigger [Guard] / Aktivität

Transition in Verhaltenszustandsautomaten

Endzustand

[Vorbedingung] Operation / [Nachbedingung]

Transition in Protokollzustandsautomaten

Pseudozustände

Startzustand

Entscheidung

Kreuzung

Austrittspunkt

Eintrittspunkt

Gabelung

Vereinigung

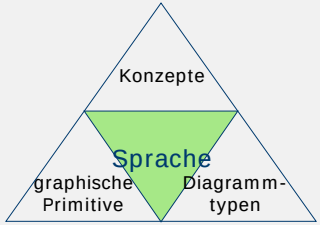
H

flache Historie

H*

tiefe Historie

Terminator



Zustandsdiagramm

sm Ampel

rot(an) /

[aus] /

rot

[reset]
schalte(rot) /

schalte(rot) /

gelb

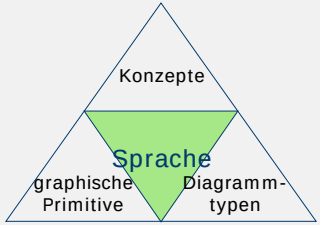
gelb(an) /

schalte(gelb) /

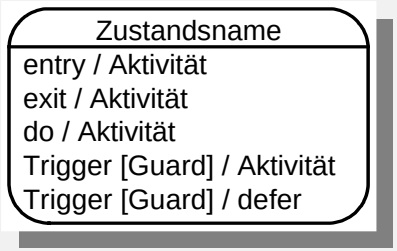
rot-gelb

grün

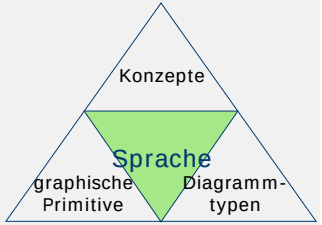
schalte(grün) /



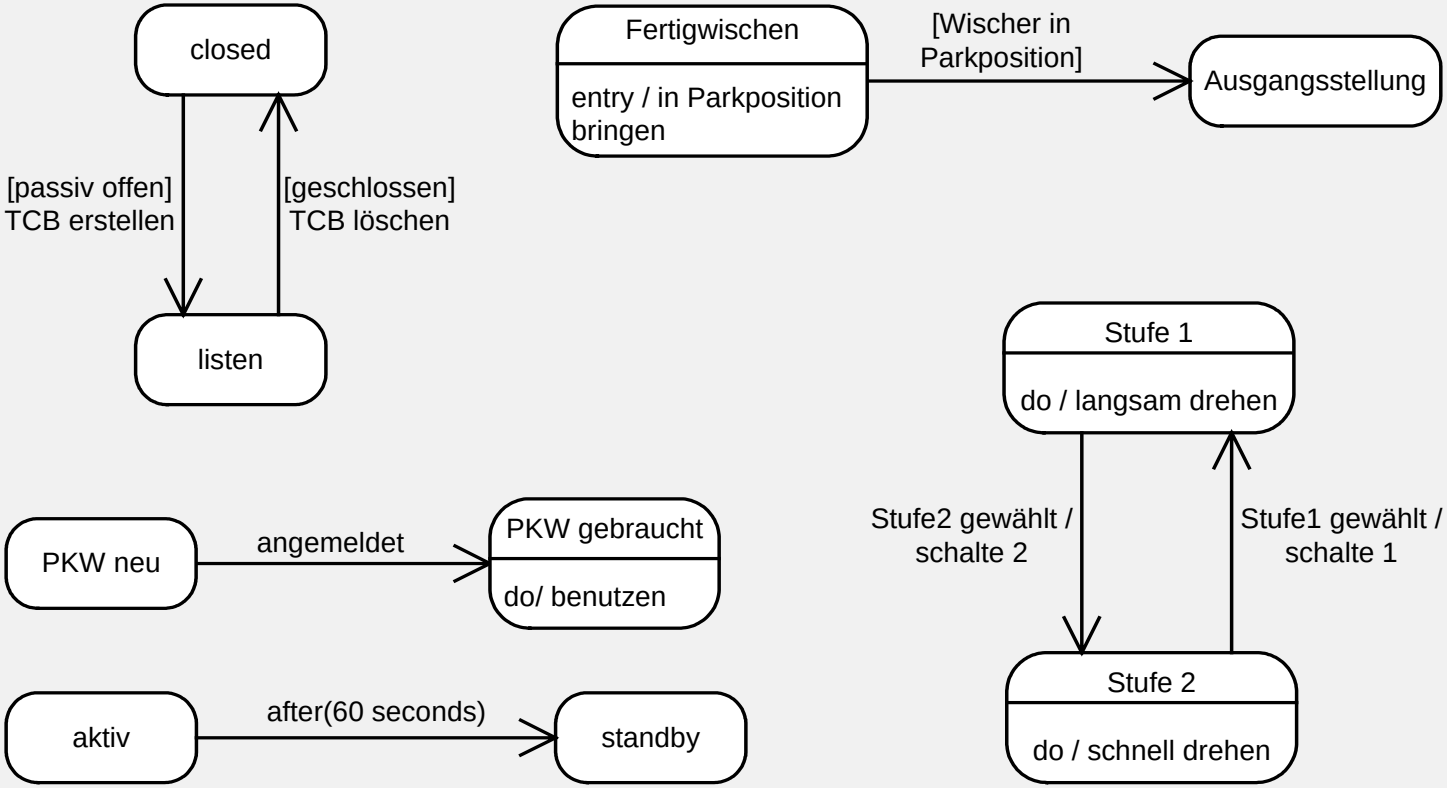
Zustandsdiagramm



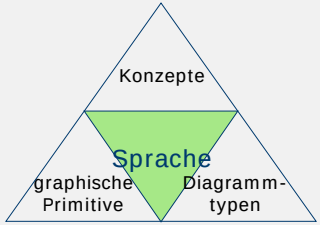
- Ein Zustand beschreibt eine bestimmte Ausprägung:
  - eine statische Situation
  - auf ein externes Ereignis wartend
- Zuständige können Aktivitäten enthalten:
  - `entry`, `do` und `exit activity`
  - verzögerte Ereignisse
  - Eine *Transition* ist der Übergang von einem Quell- in einen Zielzustand



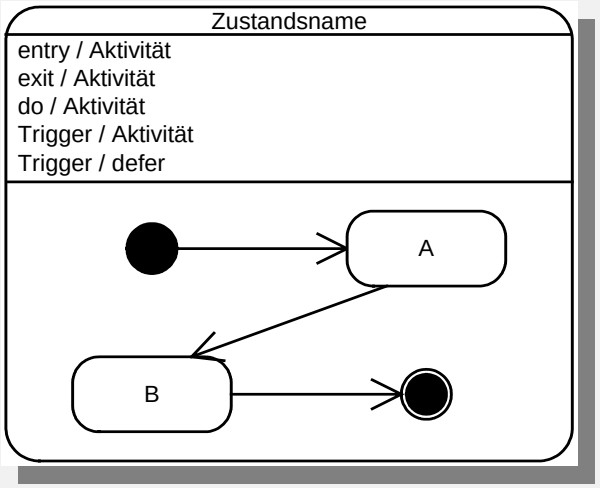
Zustandsdiagramm



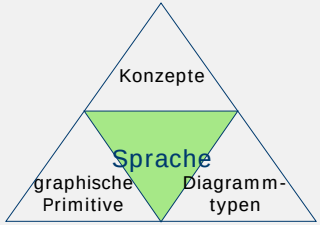
- Verschiedene Zustandsübergänge



Zustandsdiagramm



- Zusammengesetzter Zustand:
  - Setzt sich aus Zuständen, Pseudozuständen und Transitionen zusammen
  - Steht stellvertretend für einen vollständigen Zustandsautomaten
  - Kann Ein- und Austrittspunkte besitzen

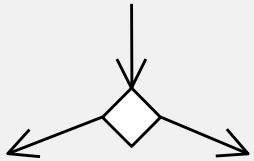


Zustandsdiagramm

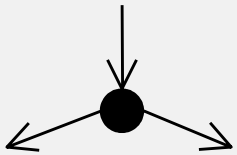
- Startzustand:
  - Verweist auf den ersten Zustand
  - Einer pro Region


Startzustand

- Entscheidung:
  - Ausgehende Transition wird während der Ausführung der Transition bestimmt


Entscheidung

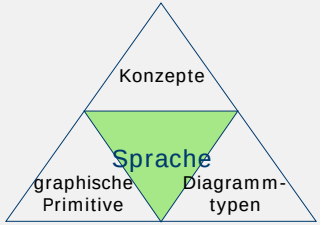
- Kreuzung:
  - Ausgehende Transition ist vor der Ausführung der Transition bekannt


Kreuzung

- Ein- und Austrittspunkt:
  - Zum Betreten und Verlassen von Unterzustandsautomaten

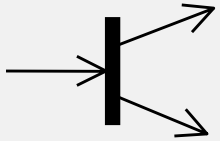

Austrittspunkt


Eintrittspunkt

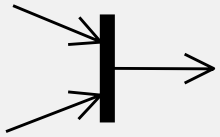


Zustandsdiagramm

- Gabelung und Vereinigung:
  - Teilen eine Transition auf mehrere parallele Zustände auf bzw. fügen mehrere Transitionen zu einer zusammen



Gabelung



Vereinigung

- Flache Historie:
  - Speichert den zuletzt aktiven Unterzustand eines komplexen Zustands

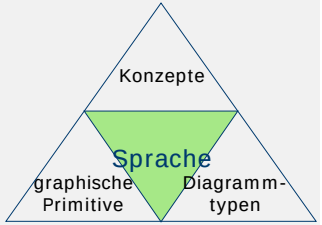
flache Historie

- Tiefe Historie:
  - Speichert den zuletzt aktiven Unterzustand eines in einem komplexen Zustand enthaltenen Zustand

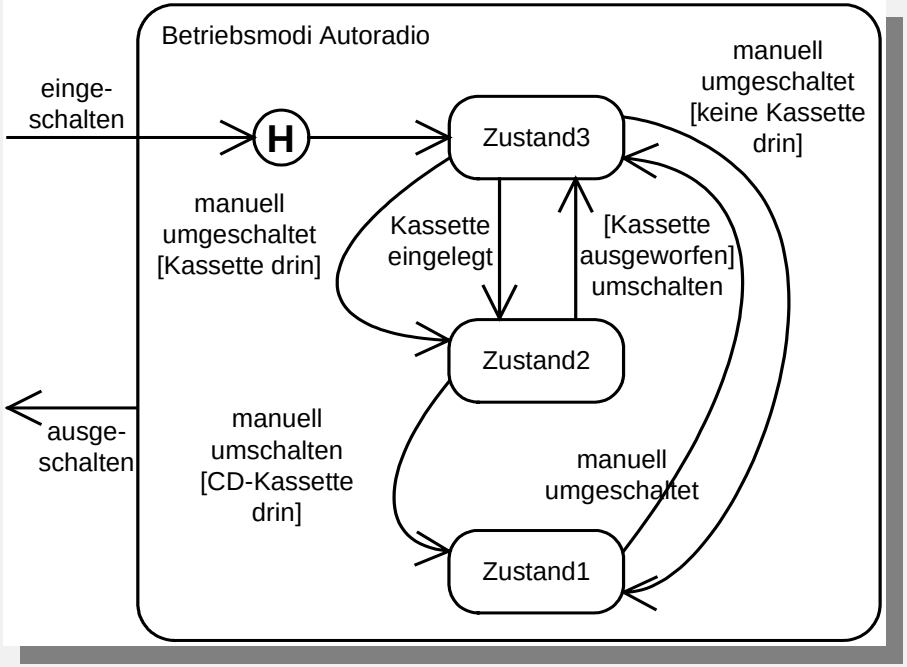
tiefe Historie

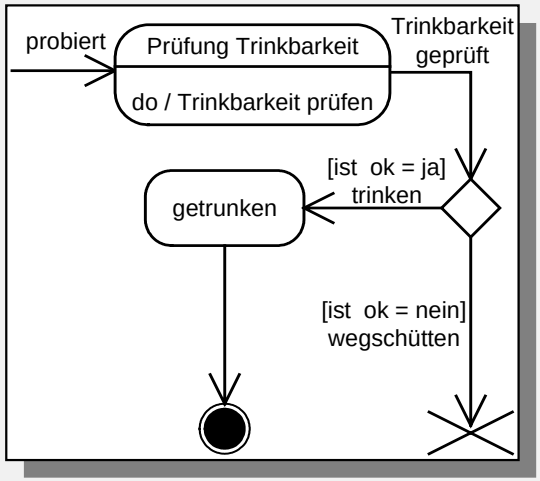
- Terminator:
  - Bei Erreichen endet die Lebensdauer der Instanz des beschriebenen Classifiers

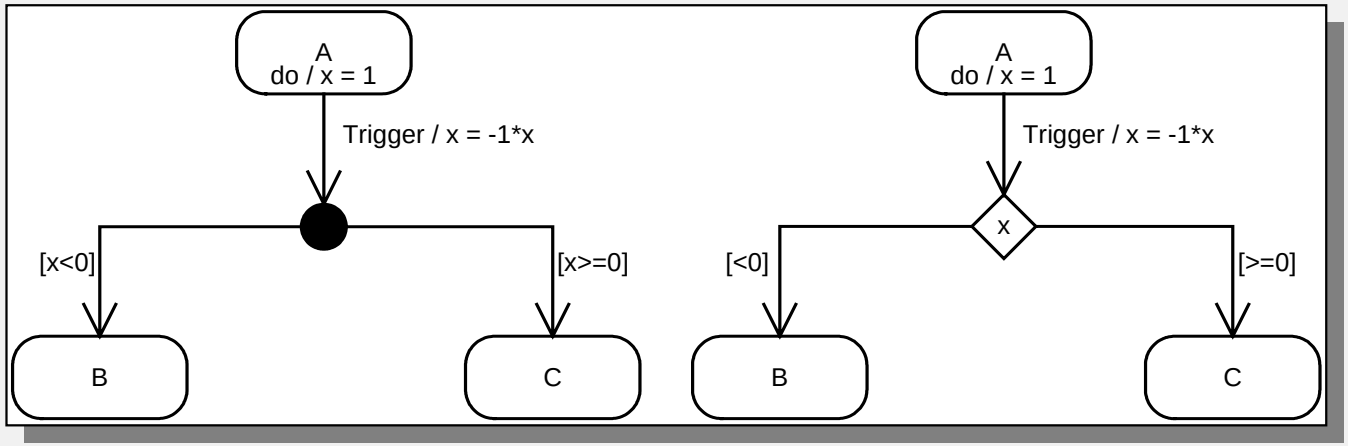
Terminator

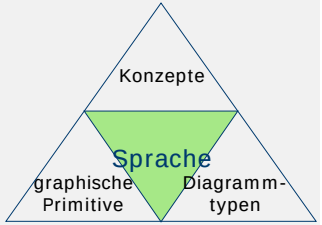


Zustandsdiagramm

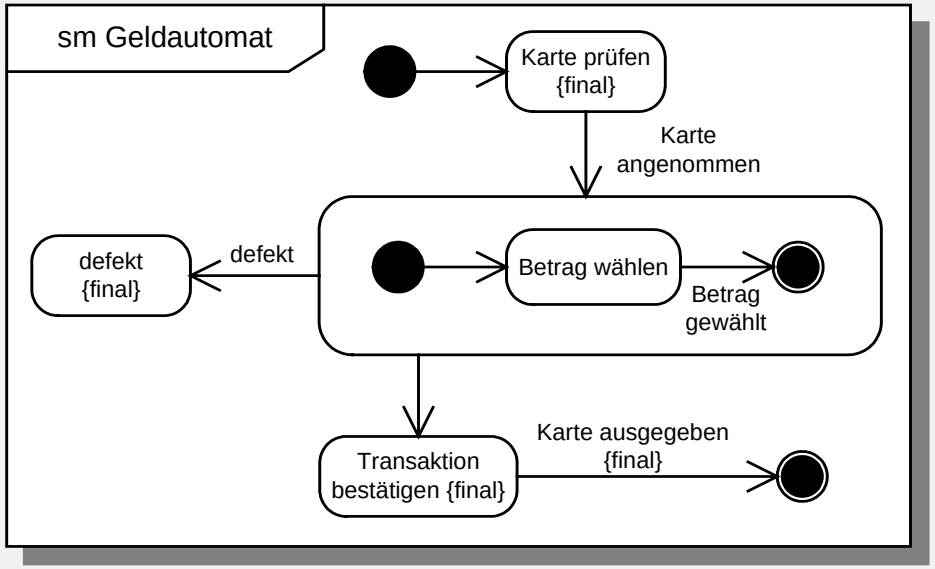


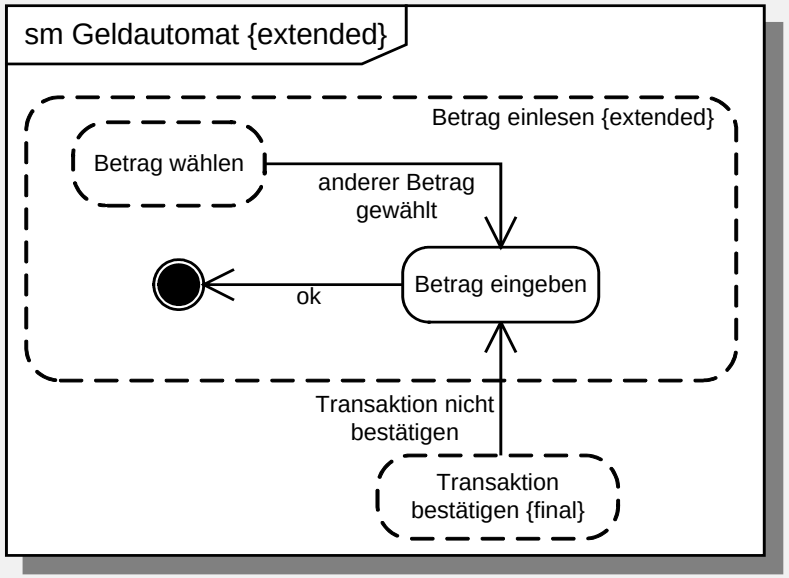




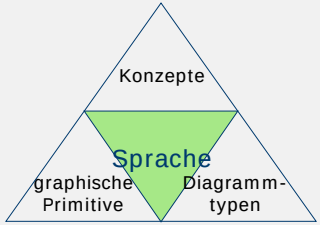


Zustandsdiagramm





- Spezialisierung von Zustandsautomaten durch
  - Erweiterung um Regionen, Zustände und Transitionen
  - Erweiterung von Regionen und Zuständen
  - Erweiterung von Transitionen



Zustandsdiagramm

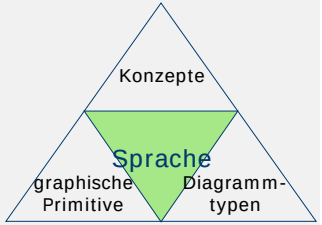
sm Port TCP active client {protocol}

closed
[aktiv offen] TCB erstellen und syn senden
[geschlossen] TCB löschen
Syn sent
[syn + ack erhalten] ack senden
established
[geschlossen] fin senden
[timeout]
fin wait 1
[fin +ack erhalten] ack senden
[ack erhalten]
fin wait 2
time wait
[ack erhalten] fin senden
aktiv geschlossen

sm Port TCP passive client {protocol}

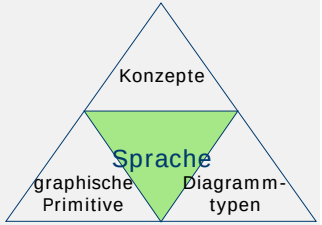
closed
[passiv offen] TCB erstellen
[geschlossen] TCB löschen
listen
[syn erhalten] syn + ack senden
[rst erhalten]
syn rcvd
[ack erhalten]
established
[fin erhalten] ack senden
passiv geschlossen
close wait
[geschlossen] ack senden
last ack
[ack erhalten]

- Protokollzustandsautomat

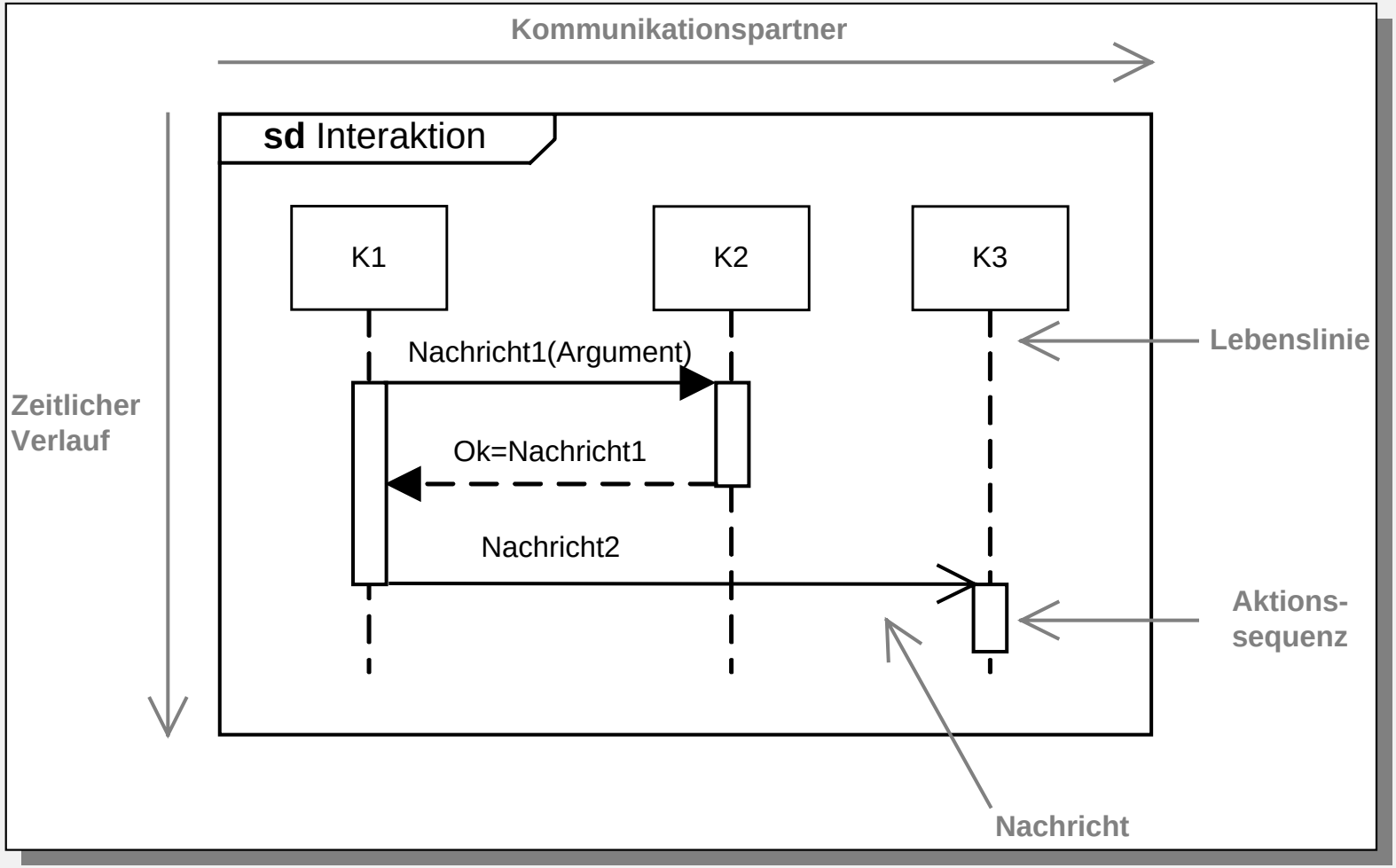


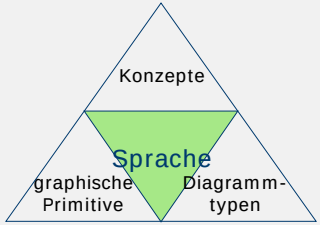
Sequenzdiagramm

- **Aufgabe**:
  - Beschreibt den Intra- und Intersystem-Datenaustausch
- **Aussage**:
  - Wie spielen die einzelnen Systeme oder –komponenten zusammen
- **Aufgabe im Projekt:**
  - Darstellung der dynamischen Aurufbeziehungen
- **Änderungen durch UML 2**:
  - Erweiterung der möglichen Kommunikationspartner
  - Referenzierung und Hierarchisierung möglich
  - Kontrollflüsse ausdrückbar
  - Neue Elemente
    - Interationsrahmen
    - Kombinierte Fragmente
    - Sprungmarken und Coregionen
    - Interaktionsreferenzen
    - *Gates* für Nachrichten



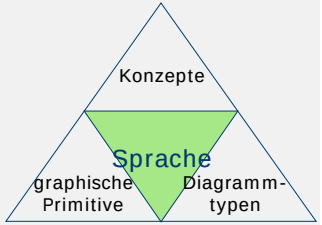
Sequenzdiagramm





Sequenzdiagramm

Lebenslinie
Zustandsbedingung
sd Bild aufhaengen
:Person
:Hammer
:Nagel
:Daumen
:Bild
[gerade]
loop
[Nagel haelt]
schlagen
getroffen = schlagen
Abbruch-bedingung
alt
treffen
Kombiniertes Fragment
treffen
Interaktions-verweis
ref
Daumen verbinden
aufhaengen
Nachricht
Aktionssequenz



Sequenzdiagramm

Synchrone Nachricht

A

B

C

N1

N3

N2

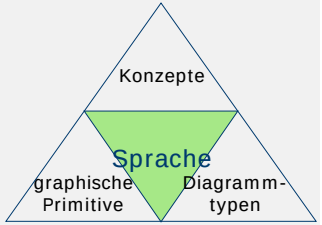
N4

Found Nachricht

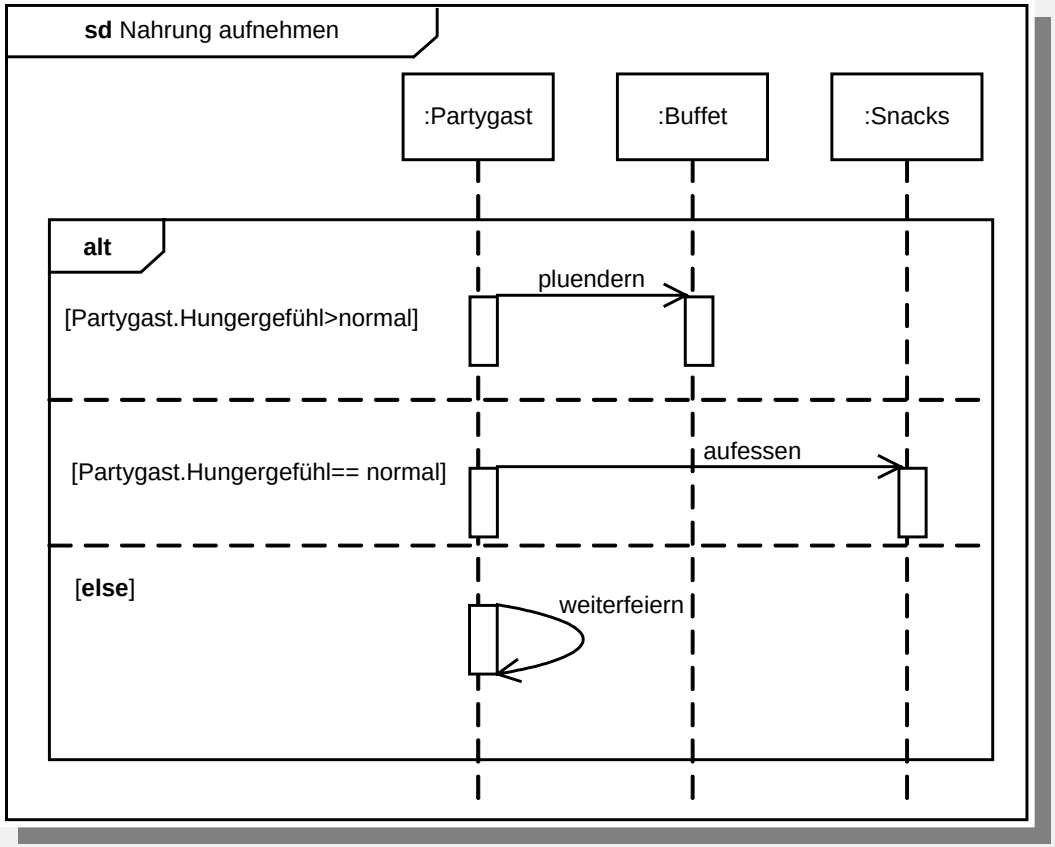
Antwortnachricht

Asynchrone Nachricht

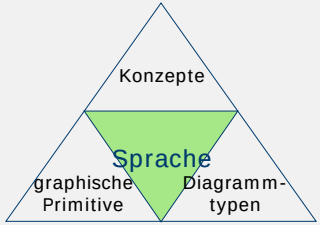
Lost Nachricht



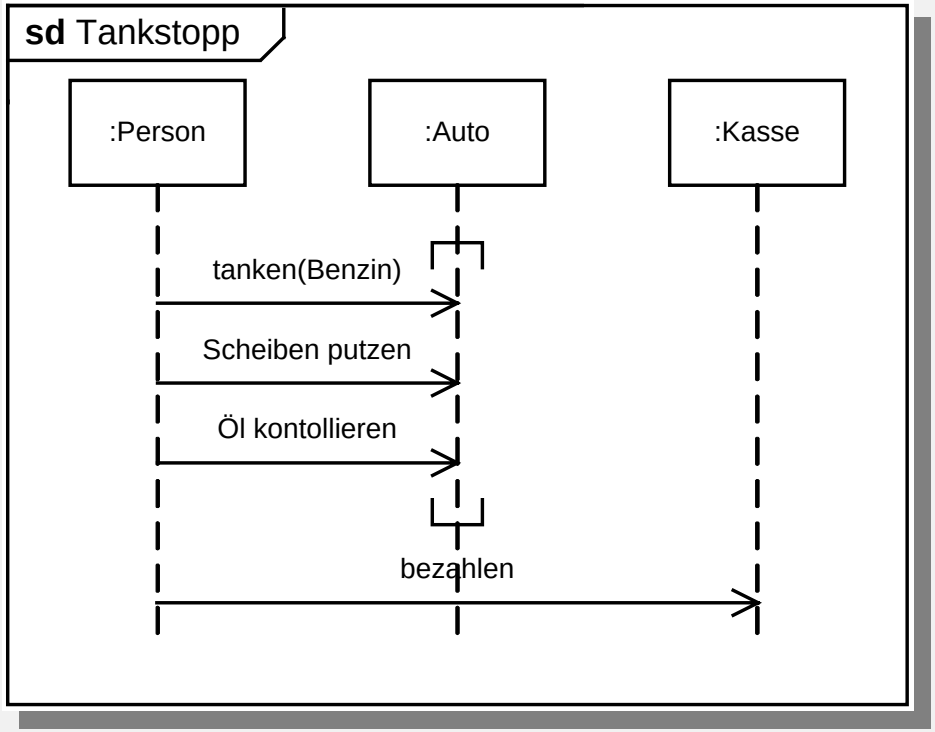
Sequenzdiagramm



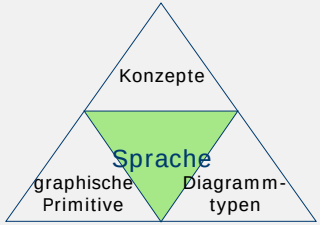
- `alt`: Bedingungsgesteuerte Alternativen (min. zwei)
- `ignore`: Gezielte Unterspezifikation (d.h. Realitätsausschnitt fehlt)
- `consider`: Betonung der Bedeutung
- `opt`: Optionale Ausführung
- `loop`: Zählschleife
- `neg`: Nicht zugelassener Ablauf
- `assert`: Zusicherung, die gelten muß
- `par`: Nebenläufigkeit oder Parallelität (wird nicht unterschieden)
- `critical`: Ununterbrechbarer kritischer Abschnitt



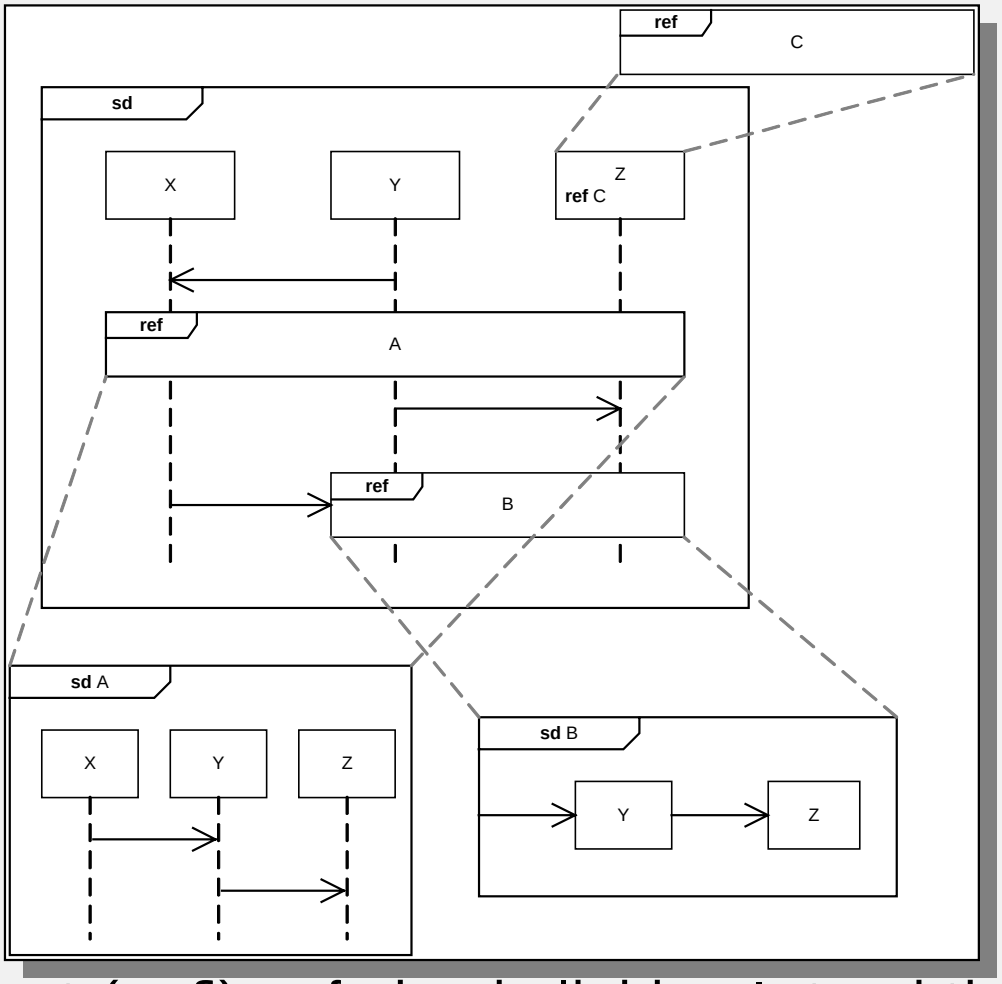
Sequenzdiagramm



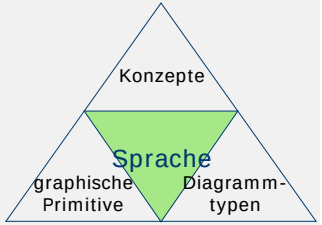
- Coregion:
  - Alternativdarstellung zum parallel kombinierten Fragment
  - Nur zugelassen wenn genau eine Lebenslinie betroffen ist
  - Ablaufreihenfolge innerhalb der Coregion nicht festgelegt



Sequenzdiagramm

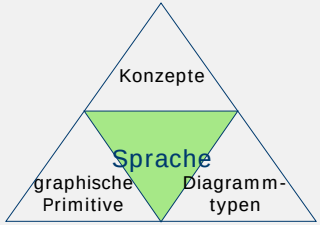


- Referenziert (ref) auf eine beliebige Interaktion
- Wiederverwendung in mehreren Diagrammen möglich
- „Zooming“–Gedanke
- Auch für Lebenslinien möglich

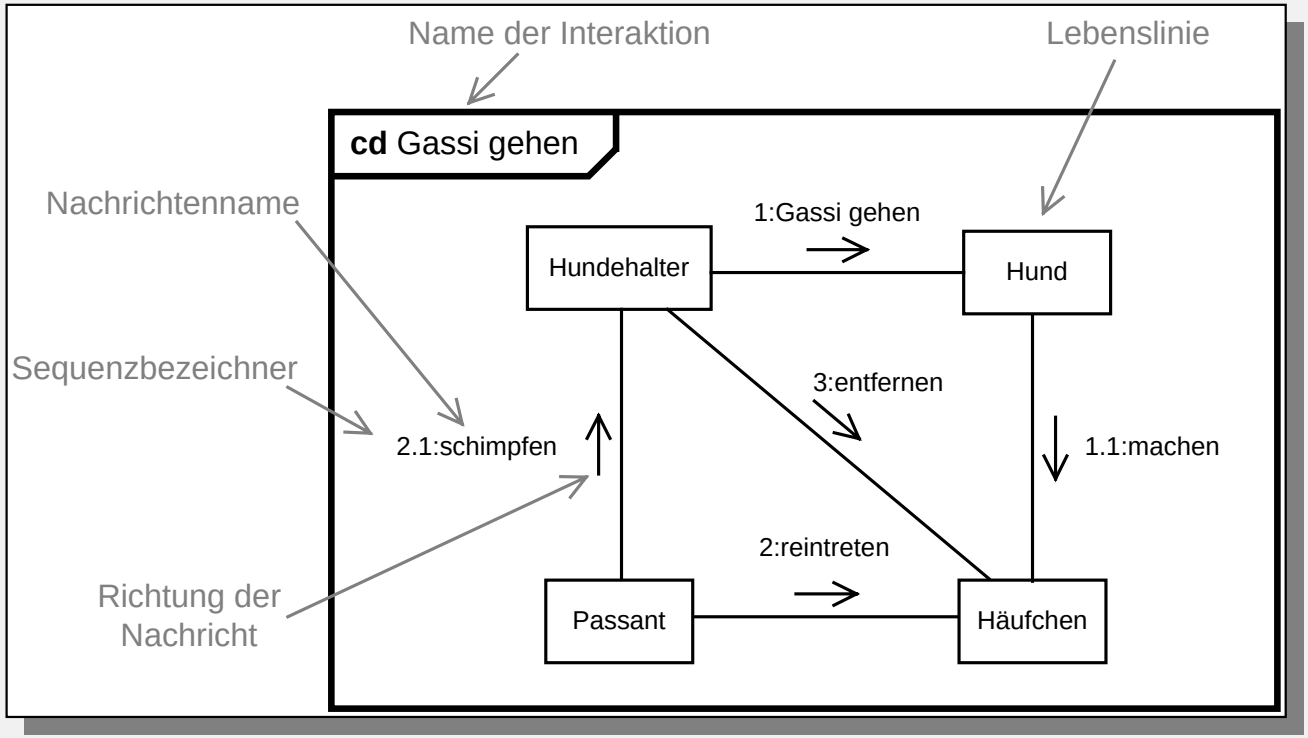


Kommunikationsdiagramm

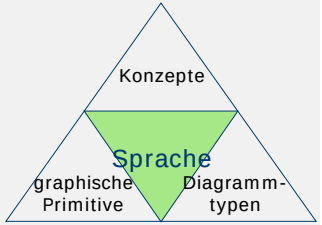
- **Aufgabe**:
  - Statische Sicht auf dynamische Interaktion
- **Aussage**:
  - Stellt Teile einer komplexen Struktur und ihre Beziehungen in der Zusammenschau dar
- **Aufgabe im Projekt:**
  - Dokumentation aller ausgetauschten Nachrichten
- **Änderungen durch UML 2**:
  - Diagrammtyp neu eingeführt
    (entspricht inhaltlich und konzeptionell dem *Kollaborationsdiagramm*)
  - Untermenge des Sequenzdiagramms
    - Keine Verweise
    - Keine kombinierten Fragmente
    - Keine Berücksichtigung der Ereignisreihenfolge



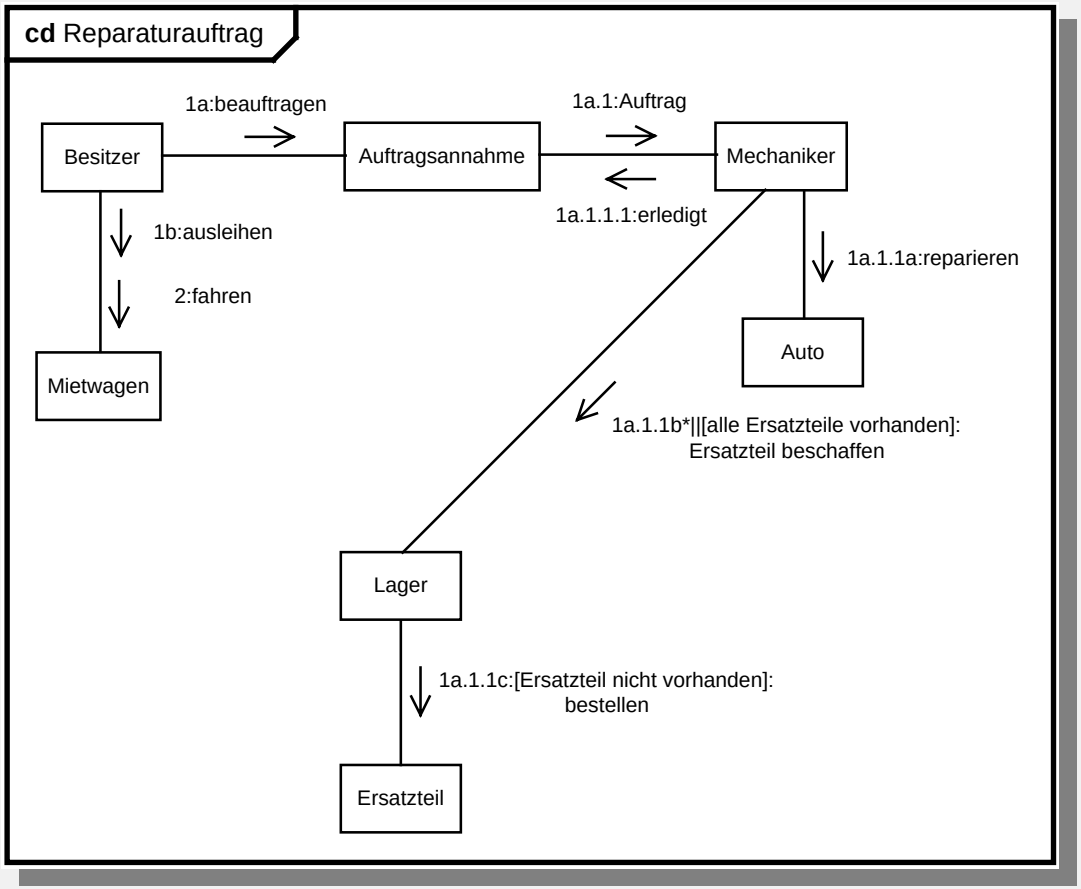
Kommunikationsdiagramm



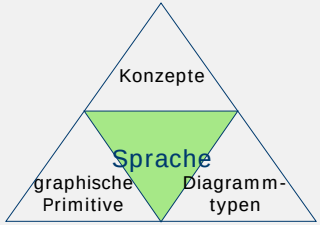
- Notationselemente:
  - Interaktion
  - Lebenslinien
  - Nachrichten
  - Sequenzbezeichner



Kommunikationsdiagramm

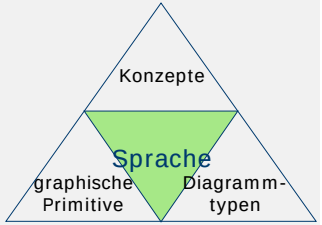


- Notation etwas unübersichtlich:
  - Nebenläufigkeit dokumentiert durch Buchstaben im Sequenzbezeichner
  - Definition von Schleifen mit einem Stern „*“
  - Kennzeichnung von nebenläufigen Schleifen-durchläufen mit Doppelstrich „||“



Timing-Diagramm

- **Aufgabe**:
  - Zeitabhängige Zustandsdarstellung
- **Aussage**:
  - Dokumentation des Zeitpunktes eines Zustandswechsels eines Kommunikationspartner
- **Aufgabe im Projekt:**
  - Dokumentation des zeitlichen (System-)Verhaltens analog einer Schaltung
- **Änderungen durch UML 2**:
  - Diagrammtyp neu eingeführt



Timing-Diagramm

Name des Diagramms

Zeit-Bedingung

td Fußgängerampel

{d...6*d}

:Ampel

grün

rot

betriebsbereit

gehen

nicht gehen

aktivieren

Lebenslinien

aktiv

:Fußgänger

wartend

d

0 10 20 30

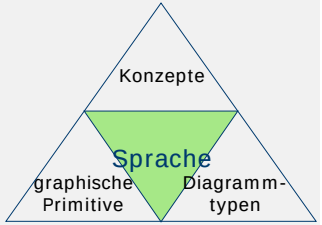
Sek.

Zustand

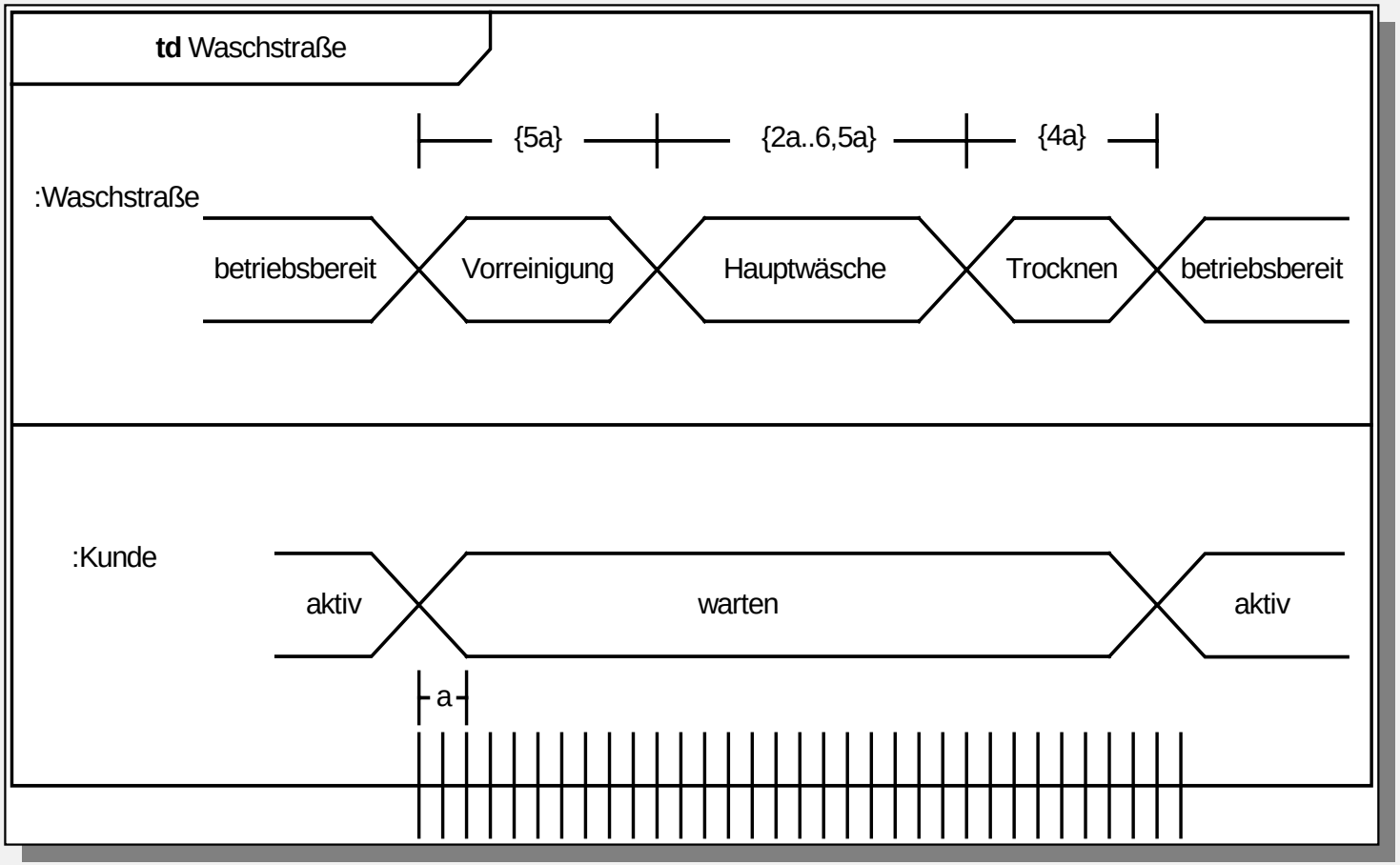
Zeitskala

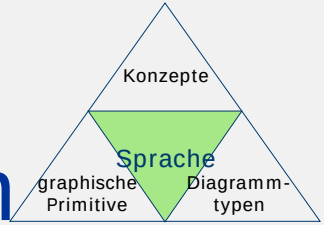
Zustandslinie

Nachricht



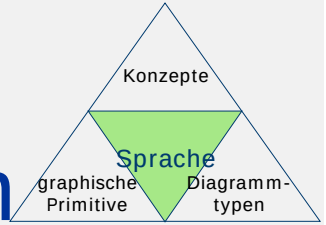
Timing-Diagramm



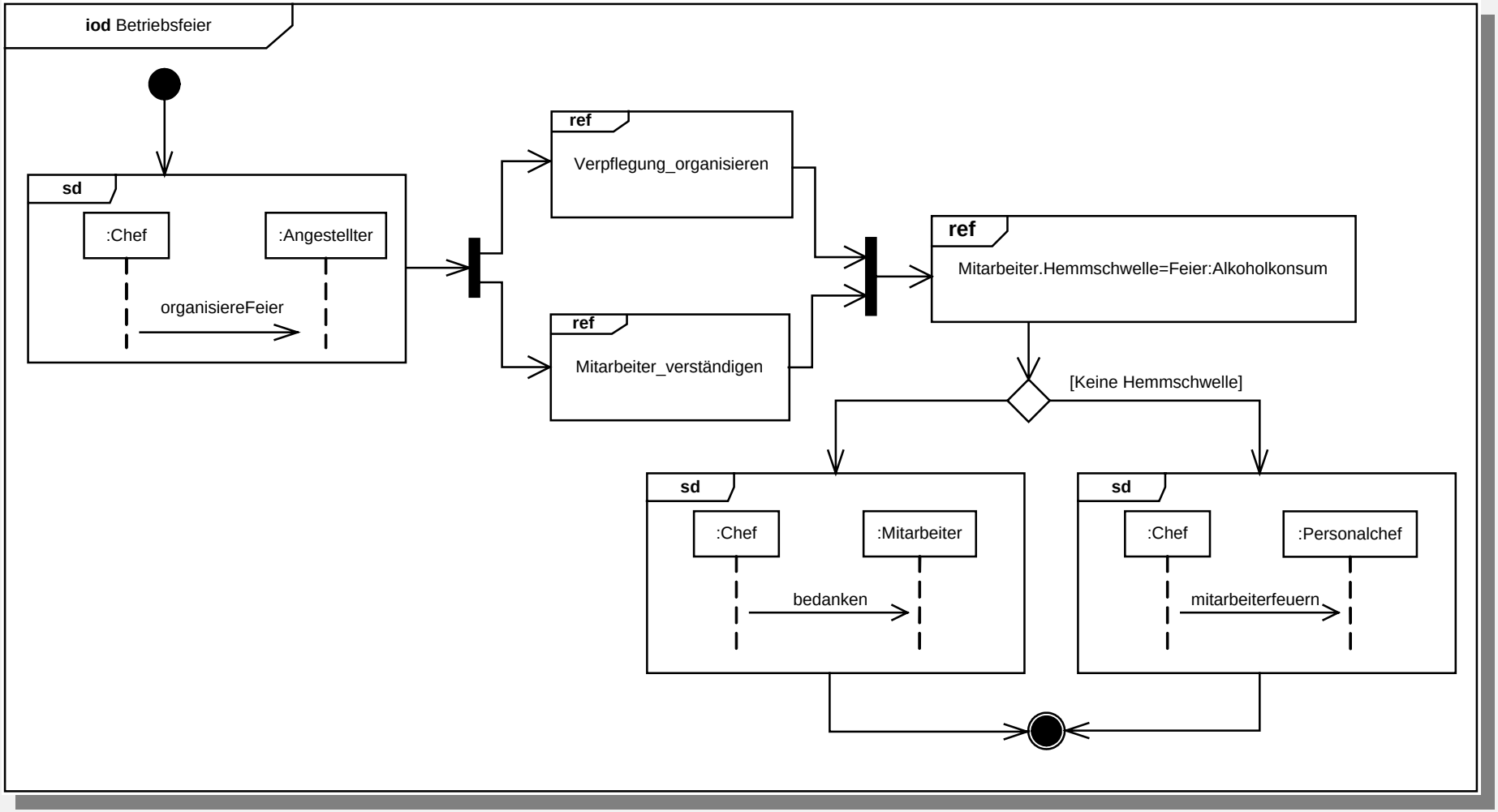


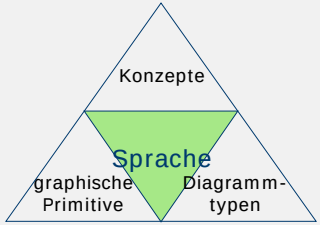
Interaktionsübersichts-Diagramm

- **Aufgabe**:
  - Darstellung des Zusammenspiels verschiedener Interaktionen
- **Aussage**:
  - Dokumentation der Bedingungen und Reihenfolgen der Interaktionsausführungen
- **Aufgabe im Projekt:**
  - Zusammenhang zwischen Aktivitätsdiagrammen
  - Übersichtlichkeitserhalt oder -gewinn
- **Änderungen durch UML 2**:
  - Diagrammtyp neu eingeführt



Interaktionsübersichts-Diagramm





Nicht-graphische Repräsentationen

- Als Alternative zu der graphischen Darstellung von Sequenzdiagrammen stehen tabellenartige Texte zur Verfügung

c11

c22

c31

⋮

c12

c21

c32

⋮

sd Polizeikontrolle

:Fahrer | :Polizist | :Alkomat | :Auto

Schlangenlinien fahren

anhalten

anhalten:ok

par

Personalien prüfen

Personalien prüfen:ok

Fahrzeugpapiere prüfen

Fahrzeugpapiere prüfen:ok

opt

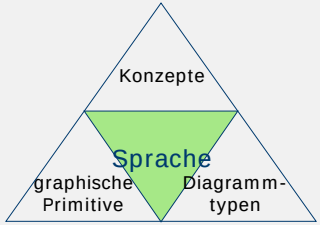
[Personalien = ok && Fahrzeugpapiere = ok]

ref :Fahrer.Alkoholwert=Alkoholkontrolle

alt

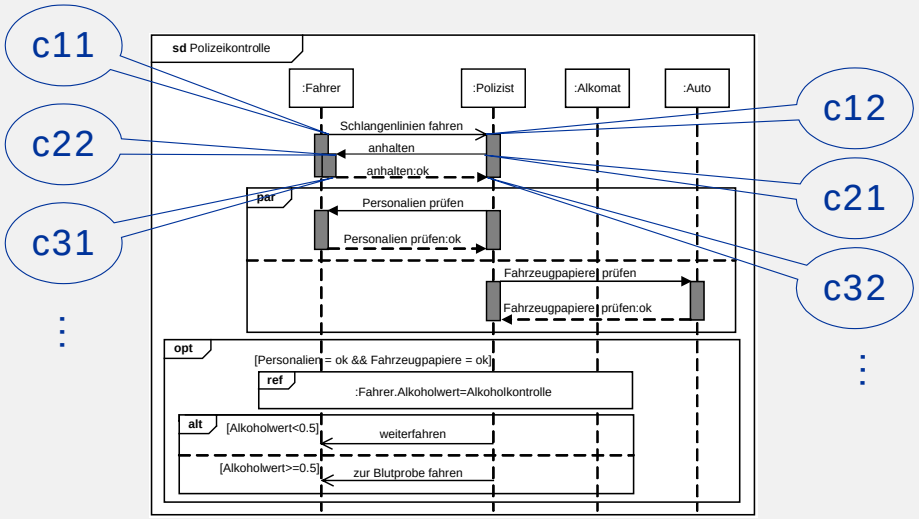
[Alkoholwert<0.5] weiterfahren

[Alkoholwert>=0.5] zur Blutprobe fahren



Nicht-graphische Repräsentationen

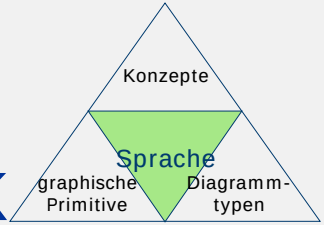
- Als Alternative zu der graphischen Darstellung von Sequenzdiagrammen stehen tabellenartige Texte zur Verfügung



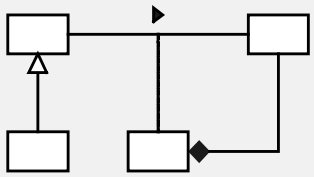
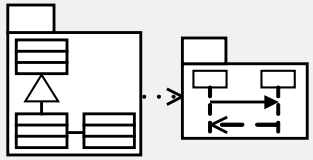
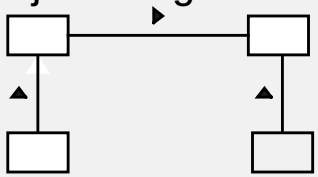
Optionale tabellarische Darstellung

- Maschinell verarbeitbar
- Möglichkeit der Konsistenzprüfung
- Übersichtlichkeit bei großen Diagrammmengen

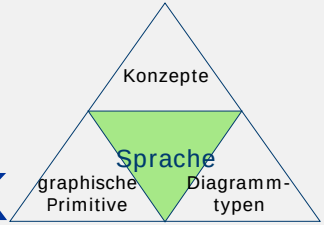
Lifeline Instance	Sending msg. Instance	Order	Msg. Name	Msg. Receive Instance
Fahrer		c11	Schlangenlinien fahren	Polizist
Polizist	Fahrer	c12	Schlangenlinien fahren	
Polizist		c21	anhalten	Fahrer
Fahrer	Polizist	c22	anhalten	
Fahrer		c31	anhalten:ok	Polizist
	Fahrer	c32	anhalten:ok	

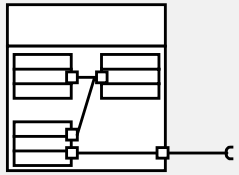
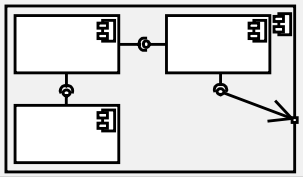
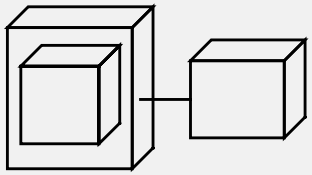


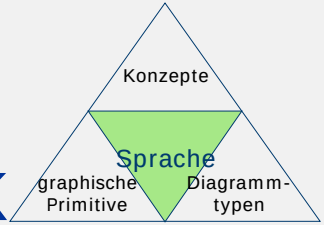
Die Diagrammtypen im Überblick

Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Klassendiagramm 	Aus welchen Klassen besteht mein System und wie stehen diese untereinander in Beziehung?	Beschreibt die statische Struktur des Systems. Enthält alle relevanten Struktur-zusammenhänge/Datentypen. Brücke zu dynamischen Diagrammen. Normalerweise unverzichtbar.
Paketdiagramm 	Wie kann ich mein Modell so schneiden, dass ich den Überblick bewahre?	Logische Zusammenfassung von Modellelementen. Modellierung von Abhängigkeiten/ Inklusion möglich.
Objektdiagramm 	Welche innere Struktur besitzt mein System zu einem bestimmten Zeitpunkt zur Laufzeit (Klassendiagramm-schnappschuss)?	Zeigt Objekte u. Attributbelegungen zu einem bestimmten Zeitpunkt. Verwendung beispielhaft zur Veranschaulichung Detailniveau wie im Klassen-diagramm. Sehr gute Darstellung von Mengenverhältnissen.

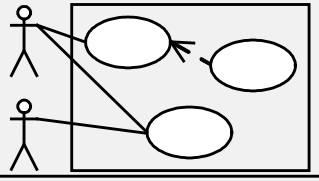
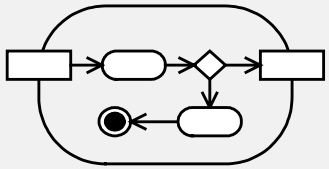
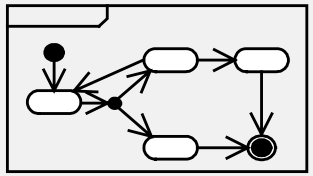
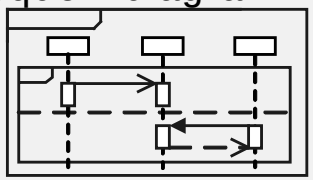
Die Diagrammtypen im Überblick



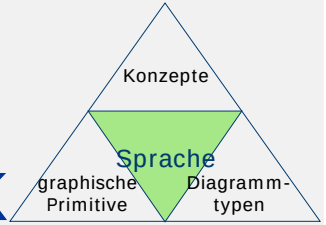
Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Kompositionsstruktur-diagramm 	Wie sieht das Innenleben einer Klasse, einer Komponente, eines Systemteils aus?	Ideal für die Top-Down-Modellierung des Systems (Ganz-Teil-Hierarchien). Zeigt Teile eines „Gesamtelements“ und deren Mengenverhältnisse. Präzise Modellierung der Teile-Beziehungen über spezielle Schnittstellen (Ports) möglich.
Komponentendiagramm 	Wie werden meine Klassen zu wieder verwendbaren, verwaltbaren Komponenten zusammengefasst und wie stehen diese in Beziehung?	Zeigt Organisation und Abhängigkeiten einzelner technischer Systemkomponenten. Modellierung angebotener und benötigter Schnittstellen möglich.
Verteilungsdiagramm 	Wie sieht das Einsatzumfeld (Hardware, Server, Datenbanken, ...) des Systems aus? Wie werden die Komponenten zur Laufzeit wohin verteilt?	Zeigt das Laufzeitumfeld des Systems mit den „greifbaren“ Systemteilen. Darstellung von „Softwareservern“ möglich. Hohes Abstraktionsniveau, kaum Notationselemente.

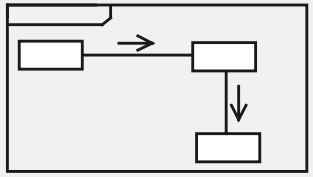
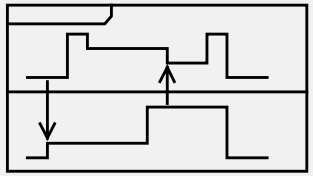
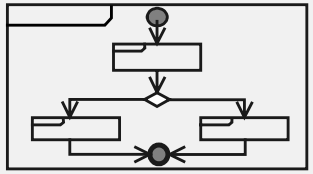


Die Diagrammtypen im Überblick

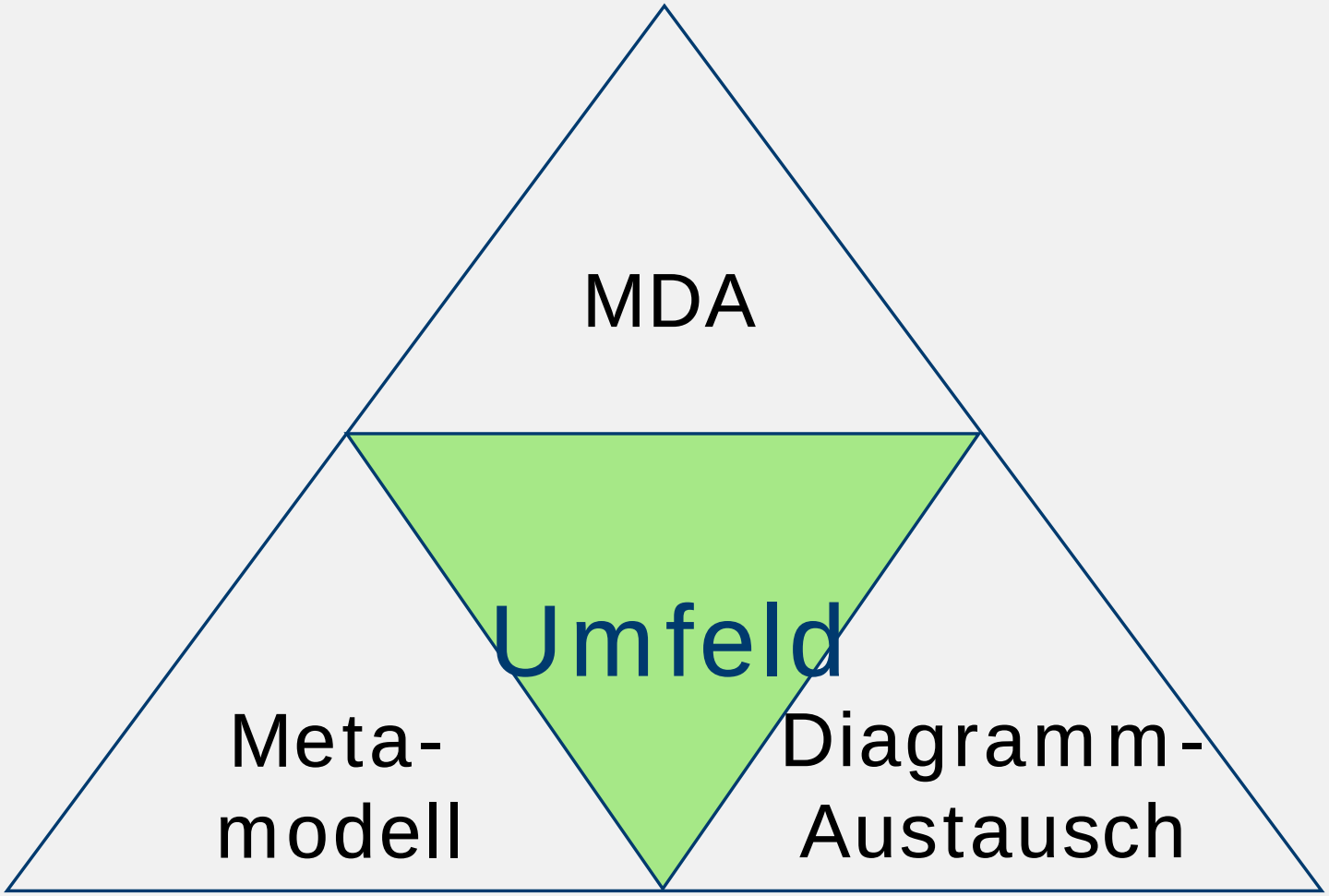
Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Use-Case-Diagramm 	Was leistet mein System für seine Umwelt (Nachbarsysteme, Stakeholder)?	Außensicht auf das System. Geeignet zur Kontextabgrenzung. Hohes Abstraktionsniveau, einfache Notationsmittel.
Aktivitätsdiagramm 	Wie läuft ein bestimmter fluss-orientierter Prozess oder ein Algorithmus ab?	Sehr detaillierte Visualisierung von Abläufen mit Bedingungen, Schleifen, Verzweigungen. Parallelisierung und Synchronisation. Darstellung von Datenflüssen.
Zustandsautomat 	Welche Zustände kann ein Objekt, eine Schnittstelle, ein Use Case, ... bei welchen Ereignissen annehmen?	Präzise Abbildung eines Zustands-modells mit Zuständen, Ereignissen, Nebenläufigkeiten, Bedingungen, Ein- und Austrittsaktionen. Schachtelung möglich.
Sequenzdiagramm 	Wer tauscht mit wem welche Informationen in welcher Reihenfolge aus?	Darstellung des Informationsaustauschs zwischen Kommunikationspartnern Sehr präzise Darstellung der zeitlichen Abfolge auch mit Nebenläufigkeiten.

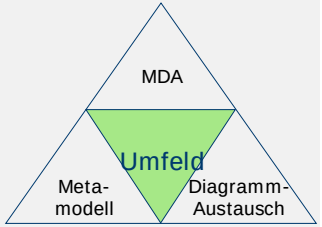
Die Diagrammtypen im Überblick



Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Kommunikations-diagramm 	Wer kommuniziert mit wem? Wer „arbeitet“ im System zusammen?	Stellt den Informationsaustausch zwischen Kommunikationspartnern dar. Überblick steht im Vordergrund (Details und zeitliche Abfolge weniger wichtig).
Timingdiagramm 	Wann befinden sich verschiedene Interaktionspartner in welchem Zustand?	Visualisiert das exakte zeitliche Verhalten von Klassen,Schnittstellen,.. Geeignet für die Detailbetrachtungen, bei denen es wichtig ist, dass ein Ereignis zum richtigen Zeitpunkt eintritt.
Interaktionsübersichts-diagramm 	Wann läuft welche Interaktion ab?	Verbindet Interaktionsdiagramme (Sequenz-, Kommunikation- und Timingdiagramme) auf Top-Level-Ebene. Hohes Abstraktionsniveau.

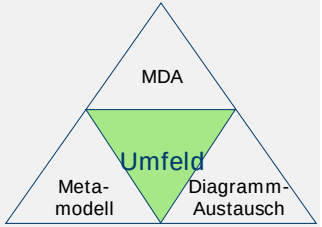
Umfeld



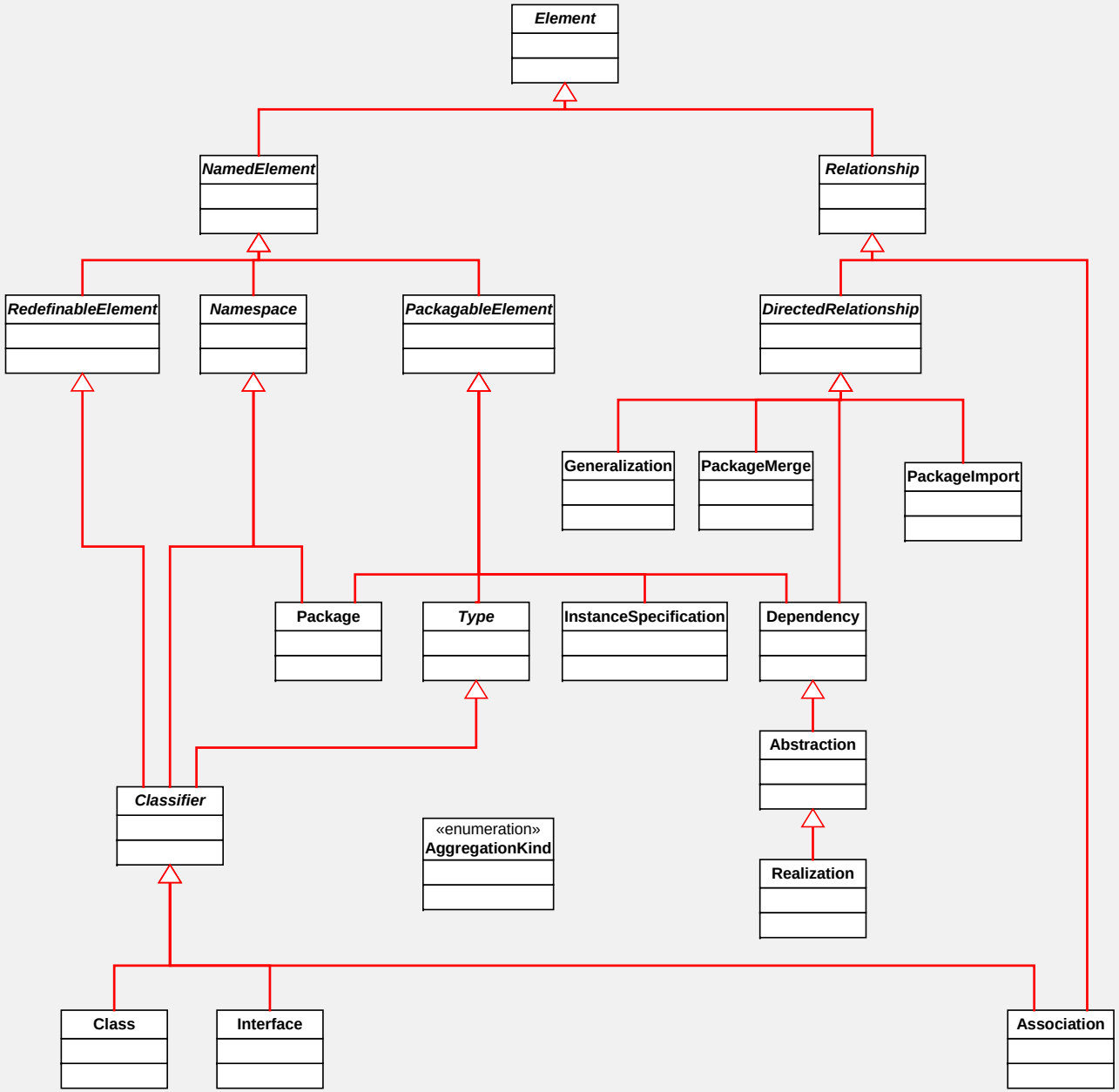


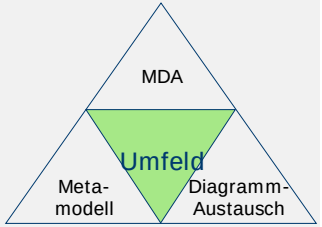
Model Driven Architecture

- Vision (weder Spezifikation noch Technik!) der OMG
- Vereinigt
  - UML (Modellierungssprache)
  - MOF (Metamodellarchitektur)
  - XMI ((Meta-)Modellaustausch)
  - Webtechniken (Web Services)
  - Generative Ansätze
- UML
  - liefert Modellierungsansatz
  - anpaßbar durch Profilbildung (integral: Stereotypen)
  - Modelle werden als *plattformunabhängig* (PIM) oder *plattformspezifisch* (PSM) rubriziert
  - Generierungsbeziehungen zwischen Modell-“Typen“

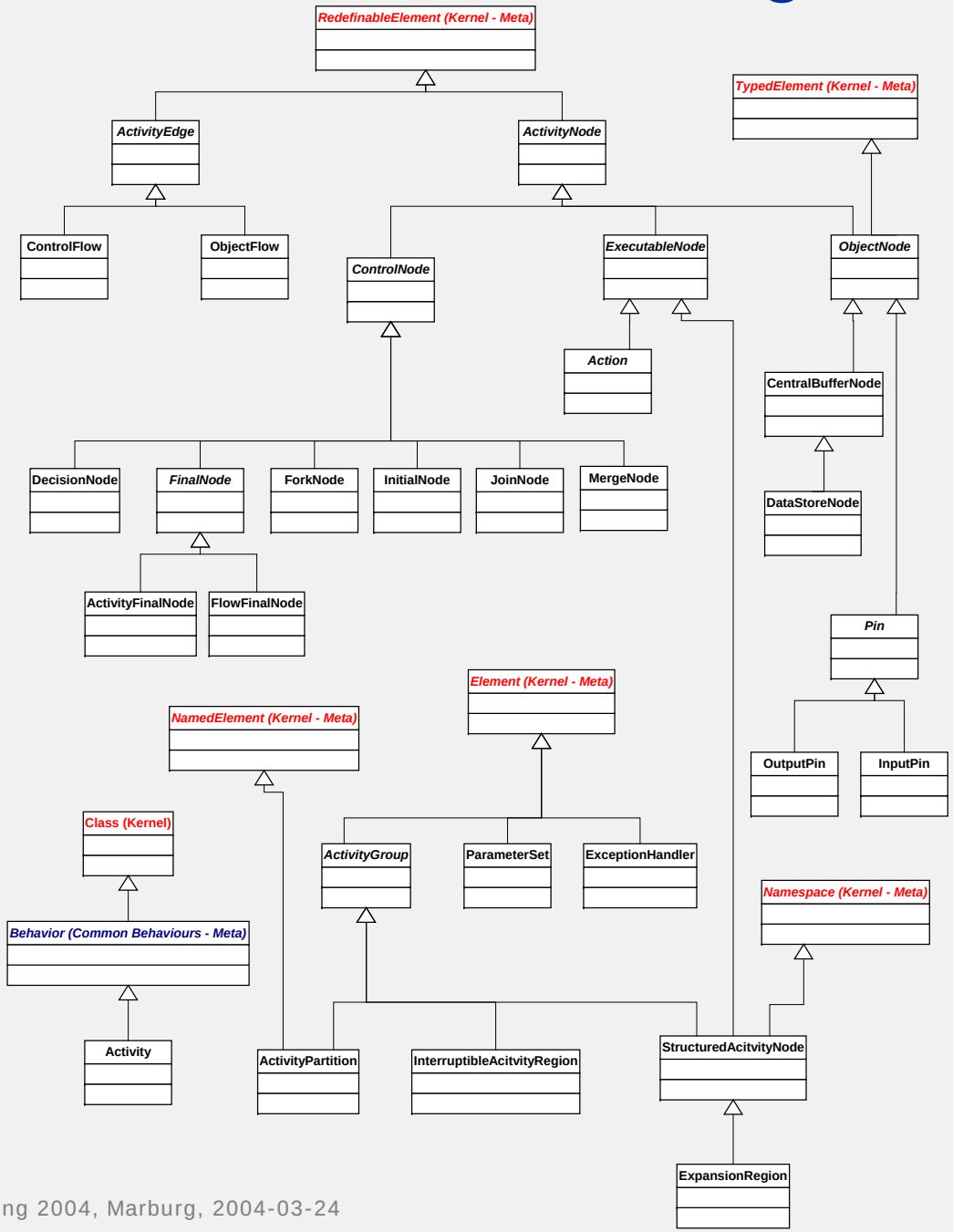


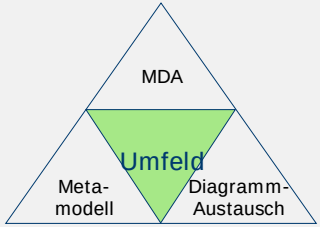
Metamodellierung





Metamodellierung

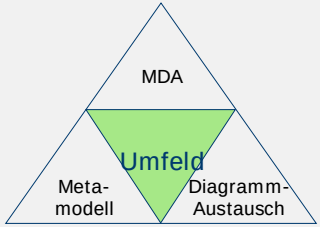




Metamodellierung

- Metamodell der UML fast vollständig überarbeitet
- Durchgängige Nutzung des Classifier-Konzeptes
- Starke Änderungen im Dynamikbereich
- Statische Modellanteile stark abstrahiert
- Erstmalige Berücksichtigung der visuellen Modellrepräsentation
- Eigenständiges Metamodell für OCL
- Erweiterungsmechanismen (Profile)

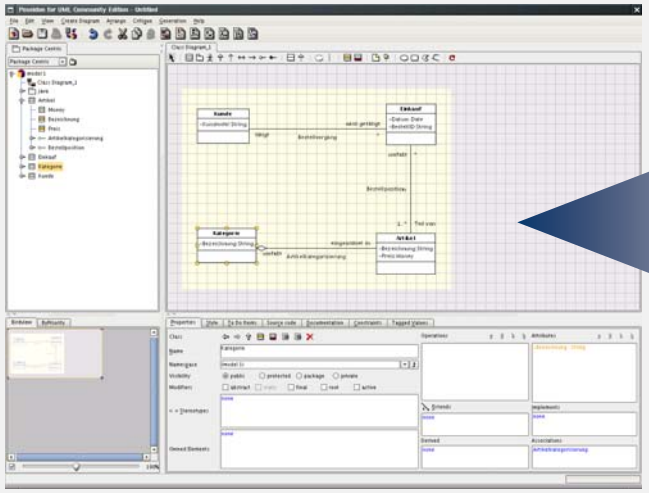
- Insgesamt:
  - Basiskonzepte grundlegend Überarbeitet
  - Verallgemeinert
  - Orthogonalität
  - Wiederverwendung (im Metamodell)
  - fast vollständig neues, kompaktes Metamodell
  - Jedoch: Einzelne Primitive (Note) bleiben immernoch völlig unberücksichtigt



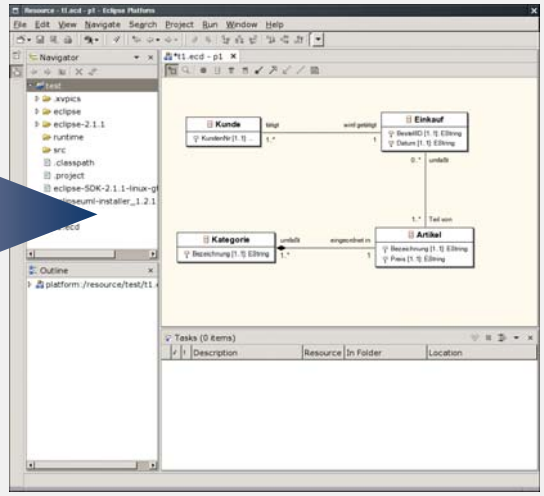
Diagrammaustausch

UML v1.x

- UML-Metamodell kennt den Begriff *Diagramms* nicht
- Modellaustausch mit dem XML Metadata Interchange (XMI)-Format als Resultat davon auch nicht.

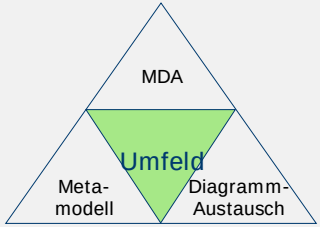


XMI
+
Diagram
Interchange



UML v2.0

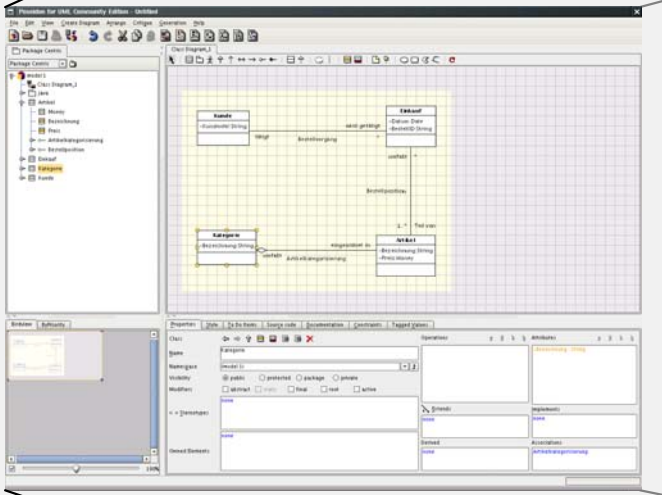
- Erweitert das bestehende Metamodell
- Enthält die Graphik*daten* (nicht die Pixeldaten)
- Kann automatisiert in andere Formate (GIF, PNG, SVG) übersetzt werden



Diagrammaustausch

UML v2.0

- Erweitert das bestehende Metamodell
- Enthält die Graphik*daten* (nicht die Pixeldaten)
- Kann automatisiert in andere Formate (GIF, PNG, SVG) übersetzt werden



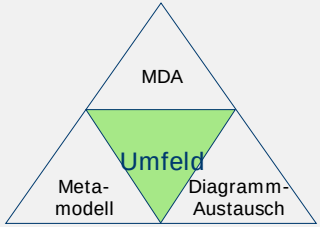
```
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp = 'Wed Dec 31 12:49:37 CET 2003'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
  </XMI.header>
  <XMI.content>
    <UML:Model xmi.id = 'lsm:aa10fc:f9cbbf26cb:-7ffa' name = 'model 1' isSpecification = 'false'
      isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
```

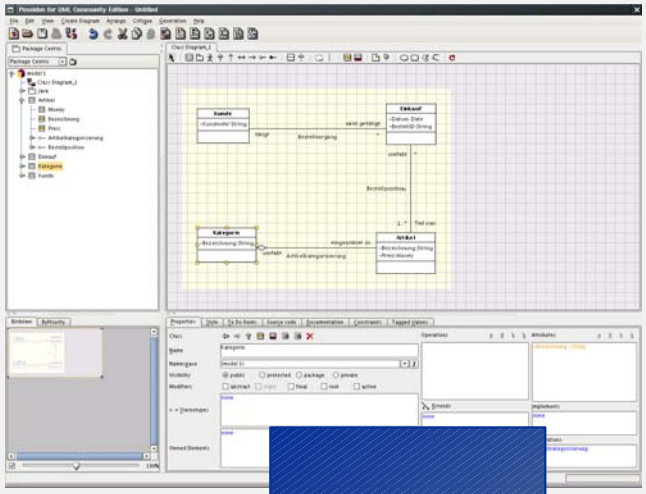
„klassische“ Modelldaten

```
...
<UML:Diagram xmi.id = 'ldi:aa10fc:f9cbbf26cb:-7fd2' isVisible = 'true' name = 'Class Diagram_1'
    zoom = '1.0'>
    <UML:GraphElement.position>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>540.0</XMI.field>
      <XMI.field>403.0</XMI.field>
    </UML:GraphNode.size>
    <UML:Diagram.viewport>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
    </UML:Diagram.viewport>
    <UML:GraphElement.semanticModel>
      <UML:SimpleSemanticModelElement xmi.id = 'lsm:aa10fc:f9cbbf26cb:-7fd1' presentation = ''
        typeInfo = 'ClassDiagram'/>
    </UML:GraphElement.semanticModel>
```

Neu mit UML 2.0: graphische Modelldaten

Diagrammaustausch



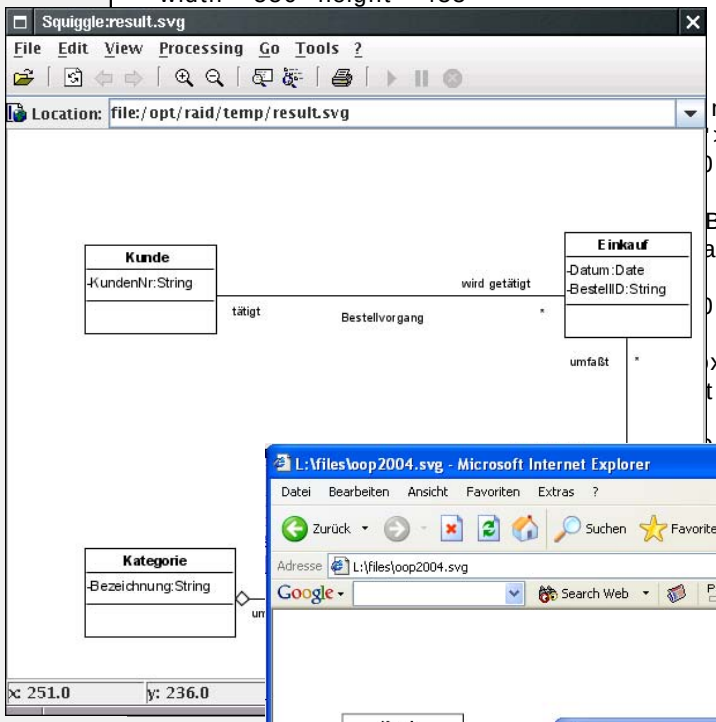


XMI
+
DI

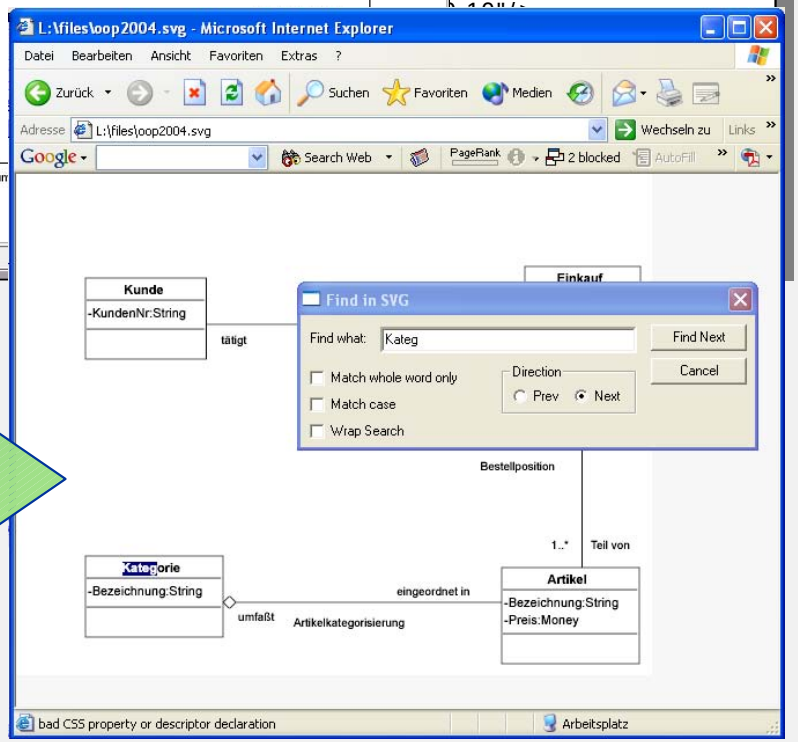

```
<xsl:stylesheet version="1.0"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:UML="org.omg.xmi.namespace.UML"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output media-type="image/svg+xml" method="xml" indent="yes"/>
    <xsl:include href="globalFunctions.xsl"/>
    <xsl:include href="templates_for_cd_dd.xsl"/>
    <xsl:param name="diagramID" />
    <xsl:param name="fillWith"/>
    <xsl:variable name="filling">
      <xsl:if test="$fillWith"><xsl:value-of select="$fillWith"/></xsl:if>
      <xsl:if test="not($fillWith)">white</xsl:if>
    </xsl:variable>

    <xsl:variable name="DiagramID">
      <xsl:if test="$diagramID"><xsl:value-of select="$diagramID"/></xsl:if>
      <xsl:if test="not($diagramID)">
        <xsl:value-of
select="//UML:Diagram[position()=1][child::UML:GraphElement.semanticModel/UML:SimpleSemanticM
odelElement[attribute::typeInfo='ClassDiagram']]/attribute::xmi.id"/>
      </xsl:if>
...
```

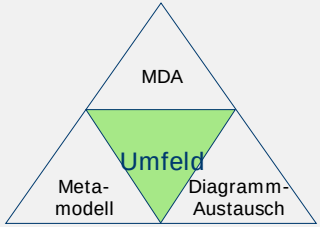
XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg"
  width="550" height="433"
```

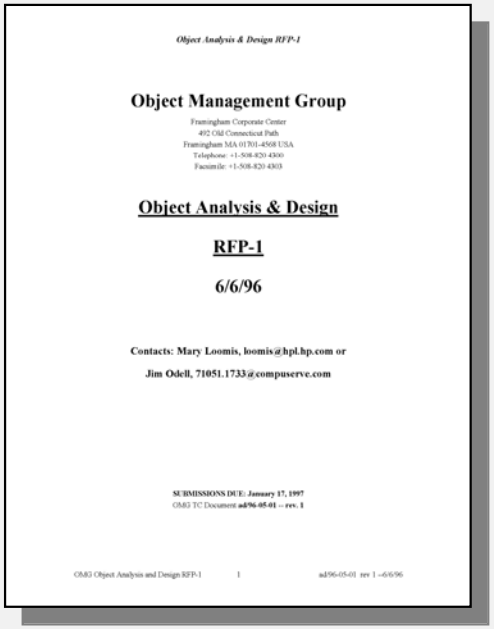



```
markerWidth="11"
>
) 10"/>

Box="0 0 10 10"
arkerHeight="11"

) 10"/>

x="0 0 10 10" refX="10"
t="11" orient="auto"
```





Standardisierung



Object Analysis & Design RFP-1

Object Management Group

Framingham Corporate Center
492 Old Connecticut Path
Framingham MA 01701-4568 USA
Telephone: +1-508-820-4300
Facsimile: +1-508-820-4303

Object Analysis & Design

RFP-1

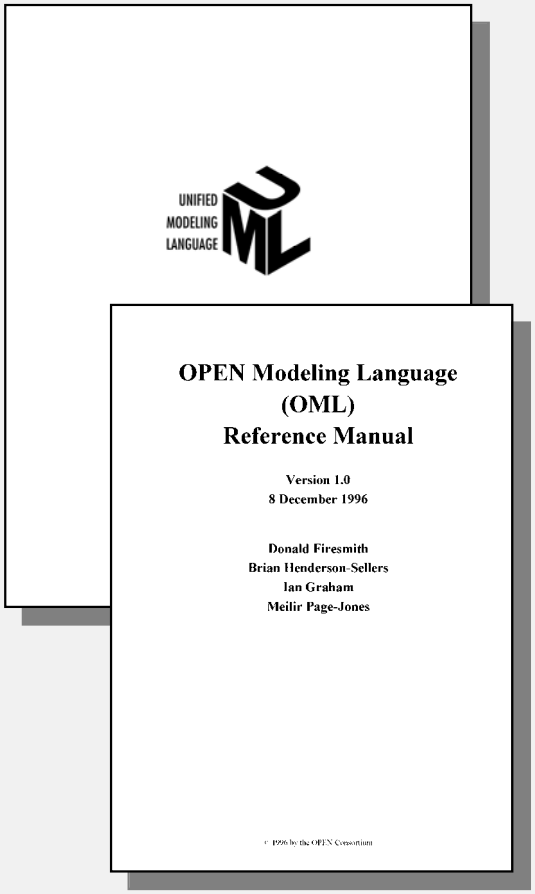
6/6/96

Contacts: Mary Loomis, loomis@hpl.hp.com or

Jim Odell, 71051.1733@compuserve.com

SUBMISSIONS DUE: January 17, 1997
OMG TC Document **ad/96-05-01 – rev. 1**

OMG Object Analysis and Design RFP-1 1 ad/96-05-01 rev 1 –6/6/96



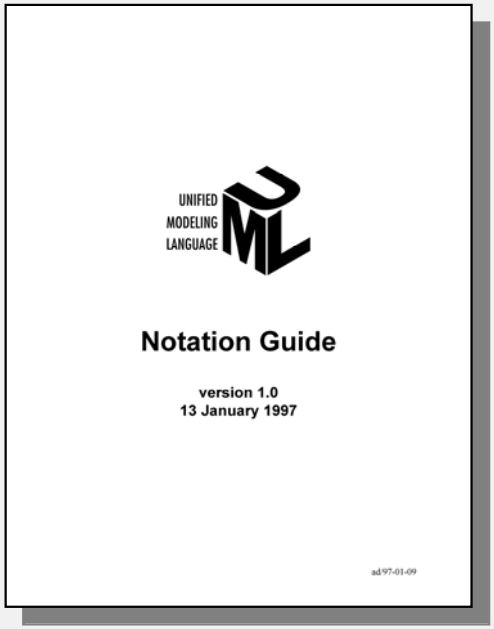
UNIFIED MODELING LANGUAGE

OPEN Modeling Language (OML) Reference Manual

Version 1.0
8 December 1996

Donald Firesmith
Brian Henderson-Sellers
Ian Graham
Meilir Page-Jones

© 1996 by the OPEN Consortium



UNIFIED MODELING LANGUAGE

Notation Guide

version 1.0
13 January 1997

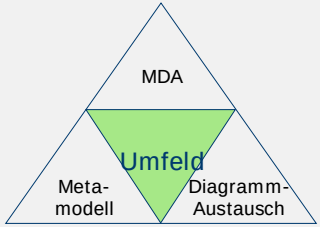
ad/97-01-09

Request for Proposals
1996

Zwei
konkurrierende
Vorschläge

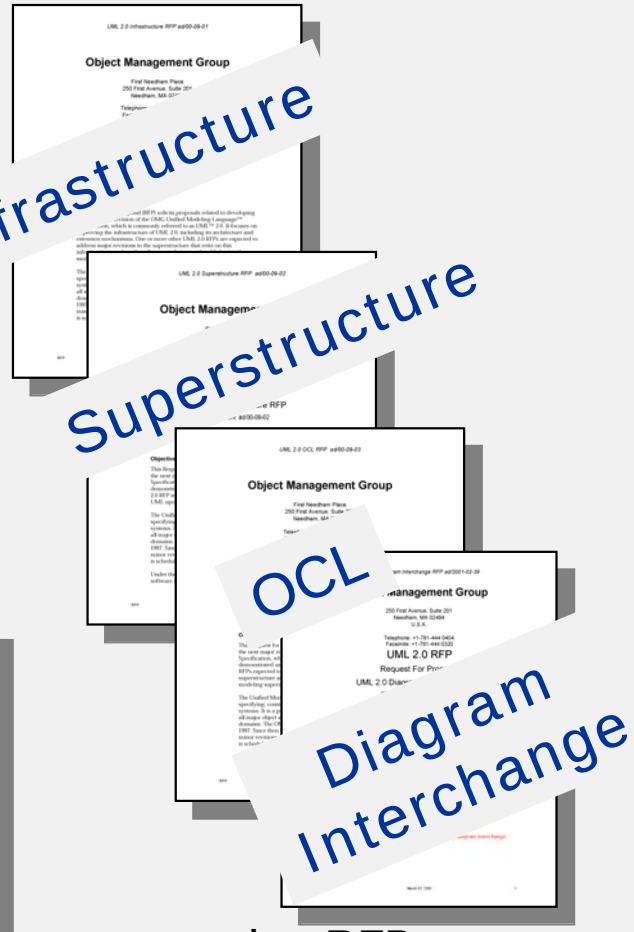
Finale Spezifikation
1997

Standardisierung

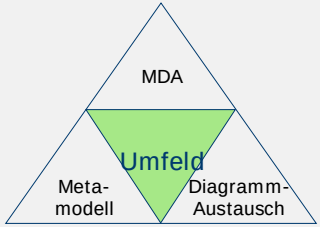


Request for Information
1999

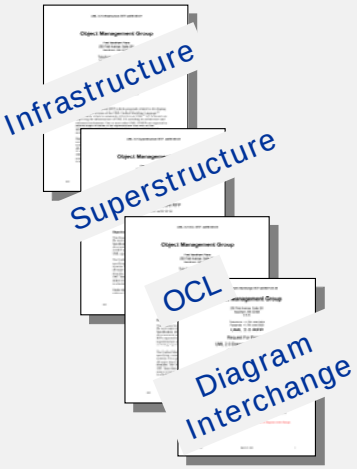
20 Einreichungen



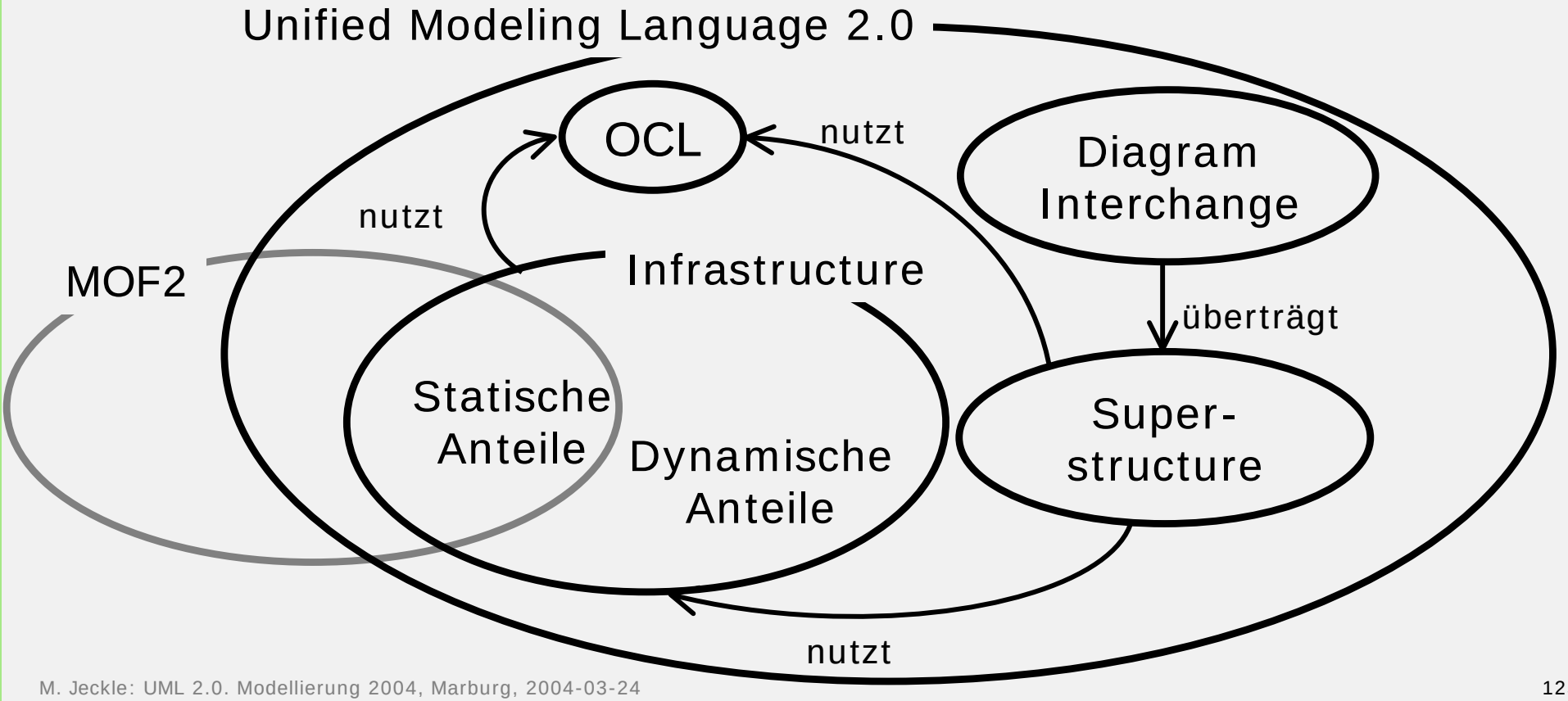
vier RFPs
2000

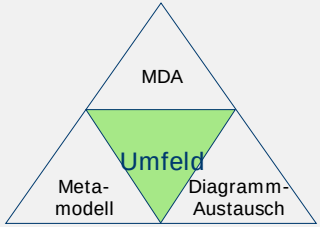


Standardisierung

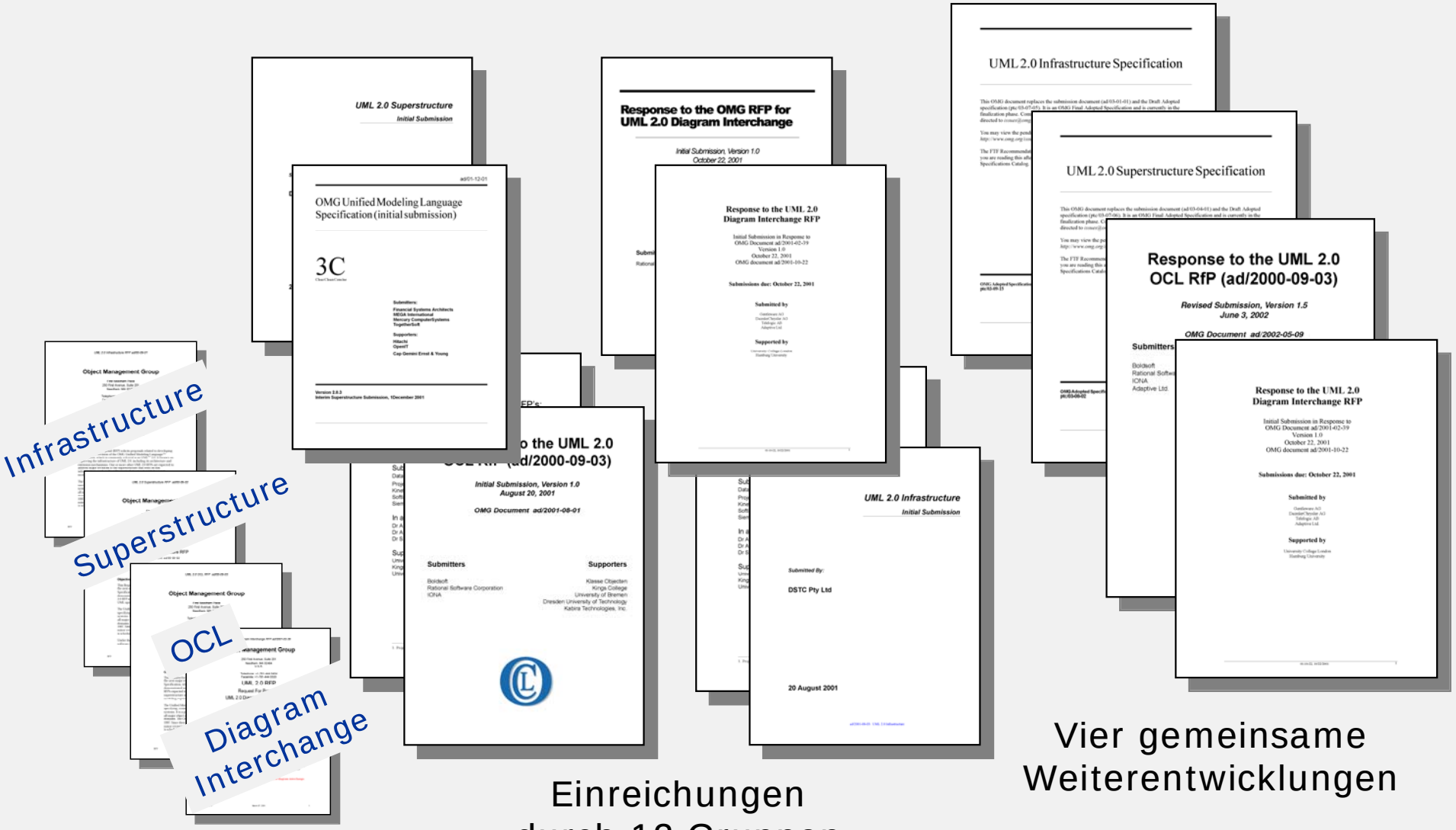


- Kein monolithischer Sprachentwurf (mehr)
- Vier separate Entwicklungsgruppen erzeugen vier separate Weiterentwicklungen mit starkem inneren Zusammenhang





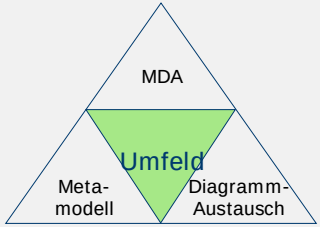
Standardisierung



vier RFPs
2000

Einreichungen
durch 12 Gruppen

Vier gemeinsame
Weiterentwicklungen



Standardisierung

- Mit UML 2.0 ist die Reise noch nicht zu Ende
- Mögliche Weiterentwicklungen
  - Vollständig ausführbare Modelle?
  - Visuelle Modelle -> Visuelle Programme?
  - Neue Paradigmen?
  - 3-D Modelle?
  - ...
- Zeitabschätzung
  - $F_{UML3} = F_{UML2} + 3 * T_{RTF} + T_{RFP}$
  - $F_{UML3} = Q2\ 2001 + (3*1) + 2$
    ≈ 2006

Inspiriert durch C. Kobryn

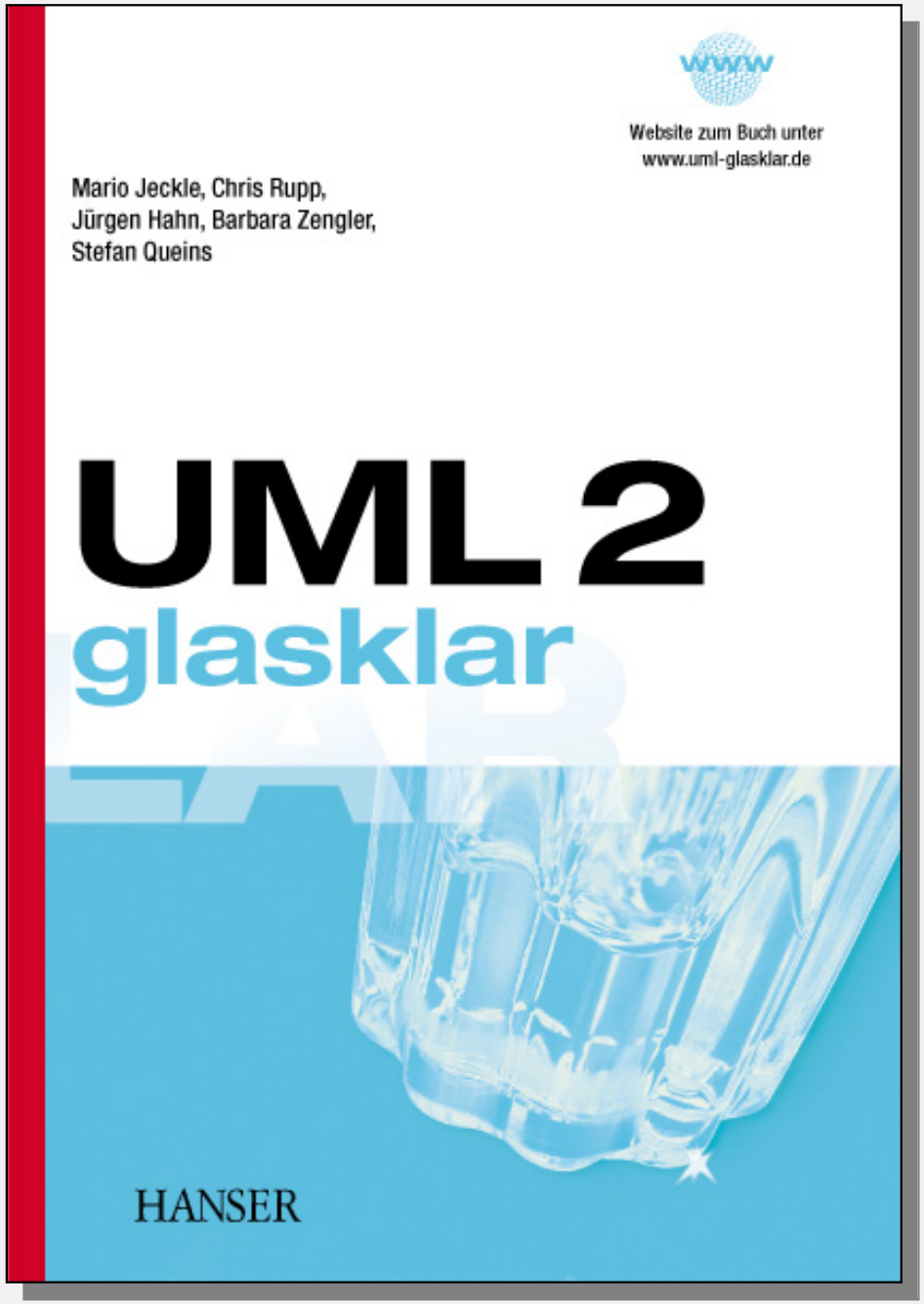
UML 2 glasklar

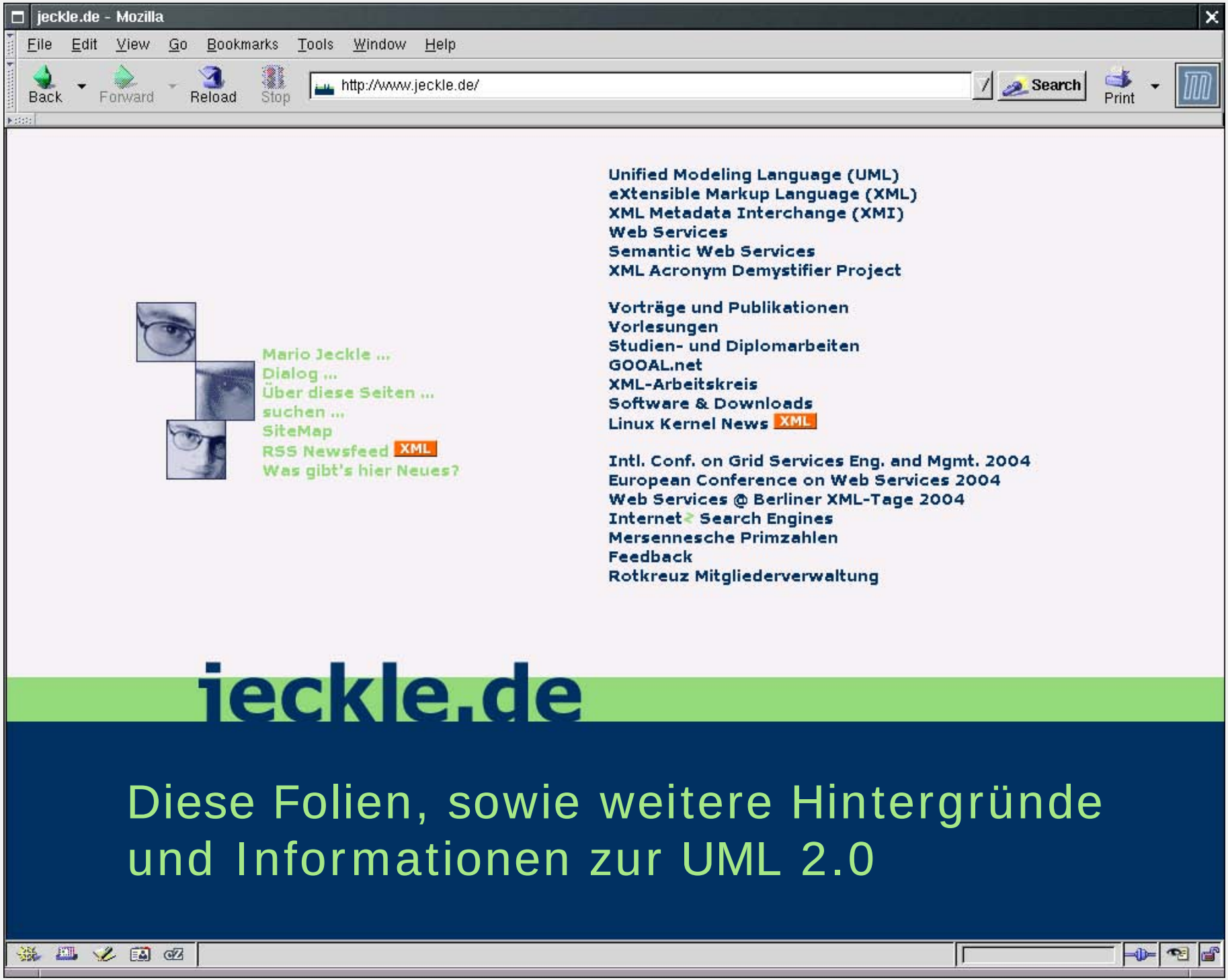
Das Buch zur Sprache

- Vollständige Behandlung der UML 2
- Alle Diagrammtypen
- Übersicht der Änderungen seit UML 1
- Grundkonzepte ohne Ballast erklärt
- Viele Anwendungsbeispiele
- Language Binding
  - C++
  - Java (incl. Generics/Java 1.5)
  - C# (incl. Generics/C# 2.0)

- Mehr Informationen:

  www.uml-glasklar.de





Diese Folien, sowie weitere Hintergründe und Informationen zur UML 2.0