LEHRSTUHL 14
SOFTWARE ENGINEERING

Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2013

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

3.0 Model-Driven Architecture (MDA)

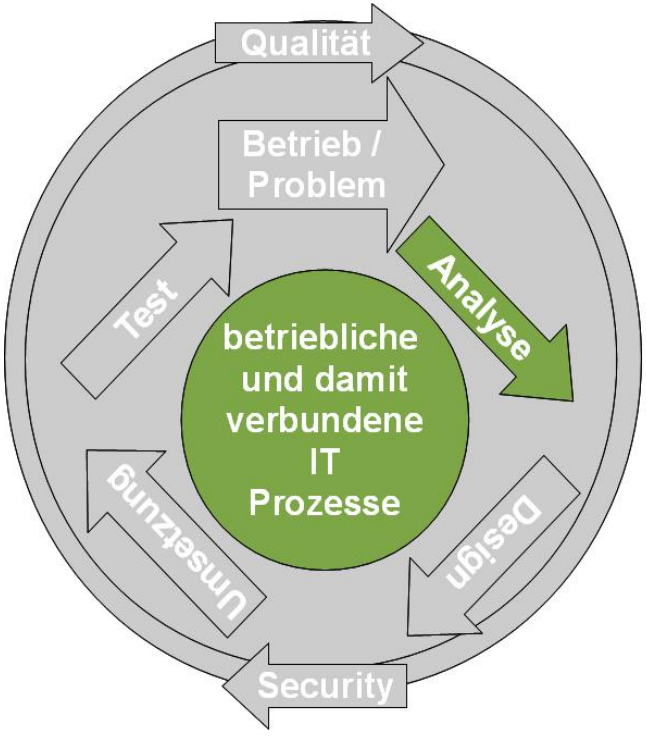
v. 08.06.2013

3.0 Model-Driven Architecture

Literatur:
Volker Gruhn, Daniel Pieper, Carsten Röttgers: **MDA - Effektives Software-Engineering mit UML 2 und Eclipse,** Springer-Verlag Berlin Heidelberg 2006
Unibibliothek (e-Book): http://www.ub.tu-dortmund.de/katalog/titel/1223129
- **Kap. 2**

Einordnung
3.0 MDA

- Geschäftsprozessmodellierung
- Process-Mining
- Modellbasierte Entwicklung sicherer Software
  - Model-Driven Architecture



Geschichte der Softwareentwicklung Anfänge

Maschinensprachen: 50er-Jahre

- Computer-Welt durch teure Hardware-Ressourcen bestimmt.
- Software hatte beherrschbare Größe; in Maschinensprache binär kodiert.

Assembler-Sprachen:

- Komplexere Software-Systeme nicht dazu geeignet, um in Binärcode entwickelt zu werden.
- Löste Maschinensprachen ab.
- Brachte mnemonische Symbole, Marken und Makros als Arbeitserleichterung.

Geschichte der Softwareentwicklung Höhere Programmiersprachen

Höhere Programmiersprachen:

- zunehmend Programmiersprachen gefragt.
- FORTRAN (FORmula TRANslator).
  - 1954-1958 von Jim Backus entwickelt.
  - Gilt als erste höhere Programmiersprache.
  - Für mathematisch-technische Probleme konzipiert.
  - Vorläuferin vieler weiterer höherer Programmiersprachen.

Geschichte der Softwareentwicklung Software-Krise

Software-Krise und Software-Engineering

- Hardware-Preise fielen.
- Bedarf an Software wuchs
- Programmierte Systeme größenmäßig so komplex und unhandhabbar.
- NATO-Konferenz (1968): Software-Krise wurde ausgerufen.
  - Forderung nach angemessenen Ingenieursdisziplin.
  - In dieser Zeit der Begriff des Software-Engineering geprägt.

Geschichte der Softwareentwicklung Software-Engineering

Strukturierte Programmierung

- Durch Kontrollstruktur- und Fallunterscheidungsmöglichkeiten.
  - z.B. etwa in Pascal oder in C

Systematische Methoden.

- Strukturierte Analyse (SA)[1]
- Strukturierte Design (SD)[2]

1Tom DeMarco: Structured Analysis and System Specification. Yourdon Press, 1978.
2 Edward Yourdon und Larry L. Constantine: Structured Design – Fundamentals of a Discipline of Computer Program and Systems Design. Prentice Hall, 1979.

Geschichte der Softwareentwicklung SA & SD

- Strukturierte Analyse:
  - Hierarchisch angeordnete Datenflussdiagramme zur abstrakten Modellierung von Prozessen.
  - Mini-Spezifikationen (z.B. in Pseudo-Code) zur Beschreibung von Prozessen.
  - Data Dictionaries für einheitlichen Datendefinitionen.
- Funktionen im Strukturiertem Design in hierarchisch aufgebaute Module zerlegt und in Strukturdiagrammen festgehalten.
- Diagramme zur Visualisierung von Programmabläufen und Beschreibung von Schnittstellen zwischen Modulen.
- Diagramme / Modelle: hoher Stellenwert in strukturierten Methoden.
- Markt für Modellierungs-Werkzeuge entstand:
  - Computer Aided Software Engineering (CASE).

Geschichte der Softwareentwicklung Wiederverwendungskrise

- Wiederverwendungskrise[1]:
- Ende der 80er-Jahre.
- Drastisch gewachsene Komplexität in Software-Systemen.
  - Erzwang Umdenken bei der Software-Erstellung.
  - Wunsch nach Wiederverwendbarkeit von Software.
- Gewaltige Software-Bestände angehäuft.
  - Problem, diese auch nur teilweise wieder zu verwenden.

Geschichte der Softwareentwicklung Objektorientiertes Paradigma

Objektorientierte Paradigma.

- Als Lösung der Wiederverwendungskrise.
- System besteht aus Objekten und jedes dieser Objekte besitzt definiertes Verhalten, einen inneren Zustand und eine eindeutige Identität.
- Ansatz in 70er-Jahren in wissenschaftlichen Veröffentlichungen erschienen.
- SIMULA (SIMUlation LAnguage)
  - Ole-Johan Dahl und Kristen Nygaard in 60er-Jahren.
  - Ursprünglich für diskrete Ereignis-Simulation entwickelt.
- Breite Akzeptanz bei Smalltalk, C++ und Java.
- Vererbung und Polymorphie für Wiederverwendbarkeit.

Design und Analyse:

- 90er-Jahre viele neue objektorientierte Methoden
  - Intensivierter Gebrauch von Modellen führte zur Unified Modeling Language (UML)

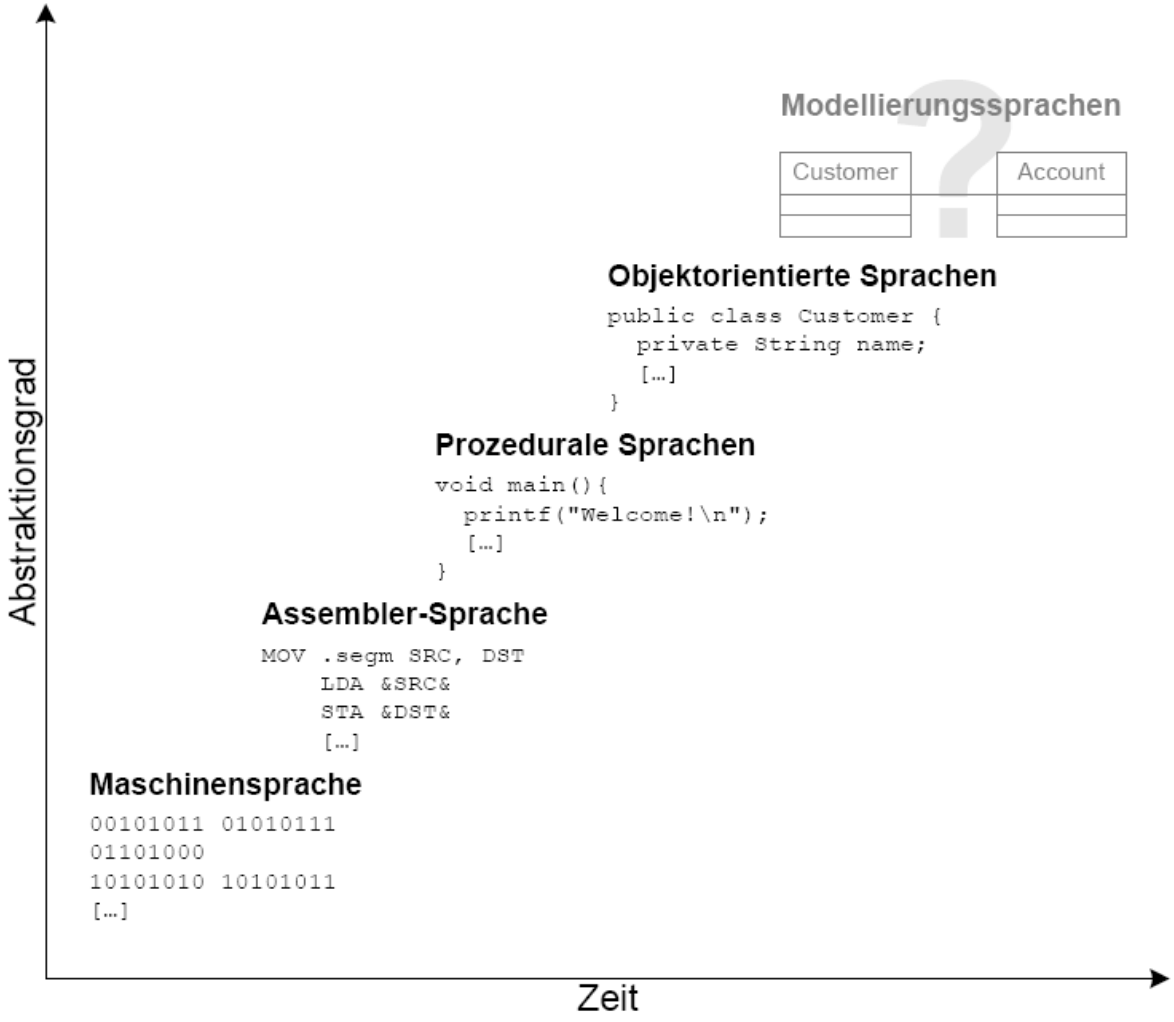
Gegenwart

Objektorientierte Paradigma weitverbreitet

Schlussfolgerungen:

- Komplexitätsbeherrschung als **das** Problem.
- Abstraktion als fortwährender Trend.
- Zunehmende Gewichtung von Modellen.

Zukunft aus bloßen Wissen geschichtlicher Entwicklung nicht vorhersagbar.



Software-Erstellung
Akute Probleme

Dominierung von Fachlichkeit durch Technik:

- Umsetzung fachlichen Basiskonzepte von Technik dominiert.
- Anwendungsentwickler benötigen umfangreiches technisches Wissen.
  - Statt sich auf fachliche Anwendungsdomäne zu konzentrieren.
- Fachabteilung und Entwickler Verständigung auf völlig unterschiedlichen Abstraktionsebenen.
  - Abstraktionsniveau derzeitiger Entwicklungsansätze zu niedrig.
- Durch Fachlichkeit bestimmten Entwicklung scheint in weiter Ferne.

Software-Erstellung
Akute Probleme

Divergenz der Änderungszyklen

- Änderungszyklus der Technologie vs. Änderungen fachlicher Prozesse (Stichwort: Globalisierung/Deregulierung).
- Technische und fachliche Belange der Entwicklung trennen.
  - Besserer Investitionsschutz der Entwicklungsarbeit
  - Verlängerte Lebensdauer der Fachkonzepte
  - Größere Flexibilität und bessere Reaktionsfähigkeit bei ändernden Fachanforderungen

Software-Erstellung
Akute Probleme

Methodischer Bruch zwischen Analyse, Design und Implementierung:

- Viele Modelle nach Beginn der Implementierungsphase nicht aktualisiert.
  - Macht Modelle nahezu nutzlos.
- Erschwert Einarbeitung neuer Mitarbeiter in späten Phasen.
- Erhöht Rüstzeiten in Betriebs- bzw. Wartungsphase um ein Vielfaches.

Fehlende Traceability:

- Fehlende Durchgängigkeit für den Lebenszyklus festzustellen
- Requirements Engineering, fehlende
  - Nachvollziehbarkeit
  - Rückverfolgbarkeit (engl. Traceability)

  von Anforderungen zu beklagen.

Software-Erstellung
Akute Probleme

Copy+Paste+Modify-Phänomen

- Gleichförmigkeit häufig übersehen oder schlicht ignoriert.

Descriptor Hell

- Middleware hat weitgehend von technischen Aspekten der Verteilung abstrahiert.
- Verteilungsflexibilität mit Zwang zu umfangreicher Konfiguration erkauft.
  - Teil der Einstellungen durch sinnvolle Vorgabewerte abgedeckt werden.
  - Deklarative Beschaffenheit legt Umsetzung in Diagrammform nahe.

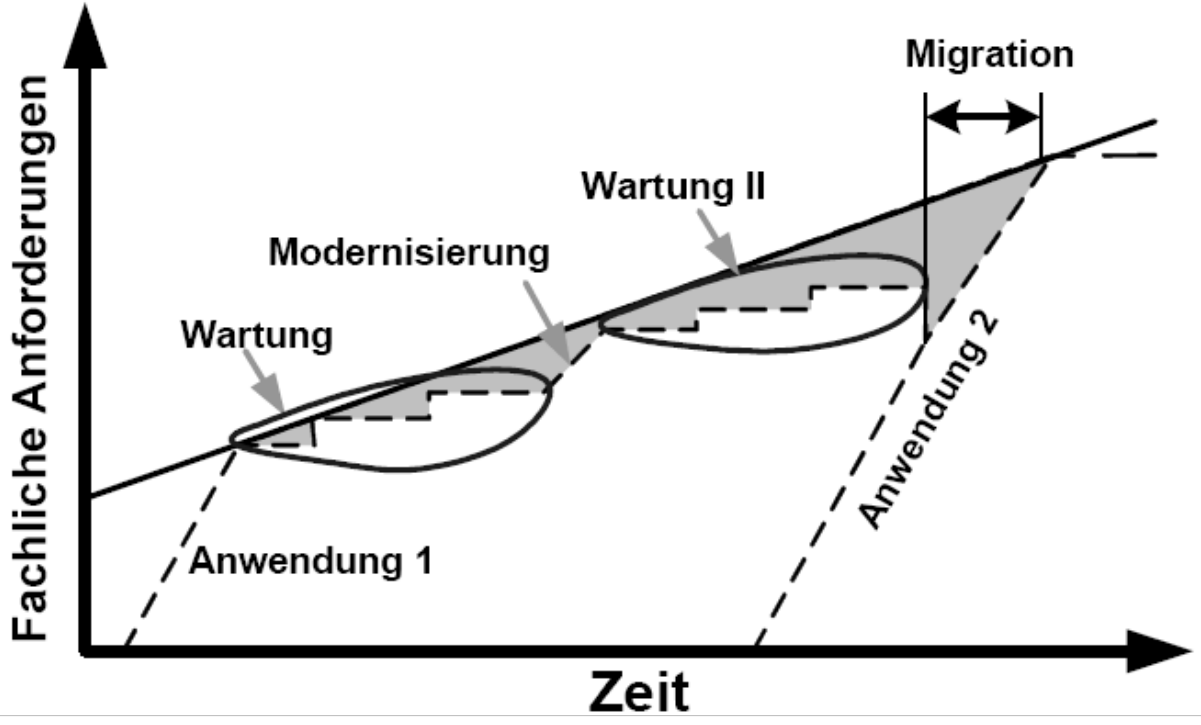
Software-Erstellung
Akute Probleme

Middleware Babel:

- Praxis ist heterogene Systemlandschaft mit vielen verschiedenen Middelware-Technologien.
- Integrationsprozess zunehmend schwieriger.

Legacy Crisis:

- Erhöhter Druck zur Modernisierung bereits bestehender Anwendungen.
- Rückstau fachlicher Änderungswünsche.
  - Bis Modernisierung notwendig wird



Akute Probleme Folgerung

- Bedarf nach höheren Abstraktionsniveau.
- Potenzial, Gleichförmigkeiten in verdichteten Form zusammenzufassen.
- Forderung nach durchgängigeren Auswahl der Ausdrucksmittel.
- Modelle:
  - Abstrahieren und fokussieren auf das Wesentliche
  - Schlagen Brücke von der fachlichen Problemwelt in die technische Lösungswelt.

Idee: Modellgetriebener Ansätze

Modellgetriebener Ansatz

- Modelle zu Artefakten erster Klasse,
  - Hinreichend zur Generierung eines Software-Systems.
- Gleich dem Übergang von deskriptiver (=beschreibender) zu präskriptiver (=vorschreibender) Verwendung von Modellen im Software-Entwicklungsprozess.
- Nicht eine universelle Entwicklungssprache für alle Problemdomänen adäquat.
- Nächsthöhere Abstraktionsstufe
  - Sprache, die lediglich dichtere Notation für Teilmenge von Konzepten gegenüber klassischen Sprachen bietet, allein nicht ausreichend.
  - Benötigen wir die Möglichkeit, domänen-spezifische Sprachen selbst zu definieren.
    - Semantische Eindeutigkeit.
    - Im Grunde wird Entwickler zum Compilerbauer

Model-Driven Architecture (MDA) Grundidee

- Spezifikation einer Softwarekomponente unabhängig von technischen Umsetzung.
- Verschiedene Abstraktionslevel für Modelle:
  - Plattform-unabhängig, als konzeptionelle Basis des Ansatzes.
  - Plattform-spezifisch.
  - Begriff der Plattform ist relativ: Spezifische Modelle einer Ebene können unabhängige Vorlage für Modelle der nächsten Stufe sein und umgekehrt.
- Übergang von abstrakteren zu technologiespezifischeren Modellen.
  - Vollautomatisiert durch Verwendung von Transformationswerkzeugen.
  - Beschreibung der Transformationen in gesondert vorzuhaltenden Transformationsbeschreibungen.

Ziele der MDA

Konservierung der Fachlichkeit:

- Wertvolles Wissen über Kerngeschäftsprozesse oft implizit in existierenden proprietären Altsystemen bzw. in den Köpfen der Entwickler.
- Plattform-unabhängige Modellierung der unterliegenden Prozesse macht es explizit und dient:
  - Pflege und Weiterentwicklung der Anwendungslogik.
  - Dokumentation und Evolution der Prozesse.
  - Integration dokumentierenden Modelle in Transformationskette hin zur Anwendung hilft gegen Degeneration von Dokumentation.

Ziele der MDA

Portierbarkeit:

- Systematische Abstraktion von technischen Aspekten erleichtert
  - Migration von Applikationen auf neue Versionen benutzter Technologien.
  - Portierung auf andere Zielumgebungen.
  - Anwendungsentwicklung für mehr als eine Zielplattform.

Systemintegration und Interoperabilität:

- Trennung fachlicher Konzepte von konkreter Repräsentation in Technologie existierender Systeme
  - Erleichtert Bildung von Schnittstellen.
- Verwendung offener Standards
  - Verringert Risiko von Vendor-Lock-Ins.
  - Hilft bestehende Anwendungen über Technologiezyklen zu retten.
- Erleichtert Wiederverwendung und steigert Produktivität.

Ziele der MDA: Effiziente Softwareentwicklung

Effiziente Softwareentwicklung:

- Werkzeugunterstützung (Automatisierung der Anwendungserstellung).
  - Beschleunigt Software-Prozesse.
  - Verringert Aufwänd.
  - Steigert Software Qualität.
    - z.B. konsequente Verwendung von Referenzarchitekturen und Pattern-Replication.

Technologien werden gekapselt.

- Wissen von Experten in Form von Transformationsvorschriften multiplizierbar.
- Ermöglicht optimale Nutzung vorhandener Ressourcen und gleichzeitig.
- Beherrschung systemimmanenter Komplexität.
  - Prinzip: Teile-und-herrsche.

Domänen-Orientierung

Domänen-Orientierung:

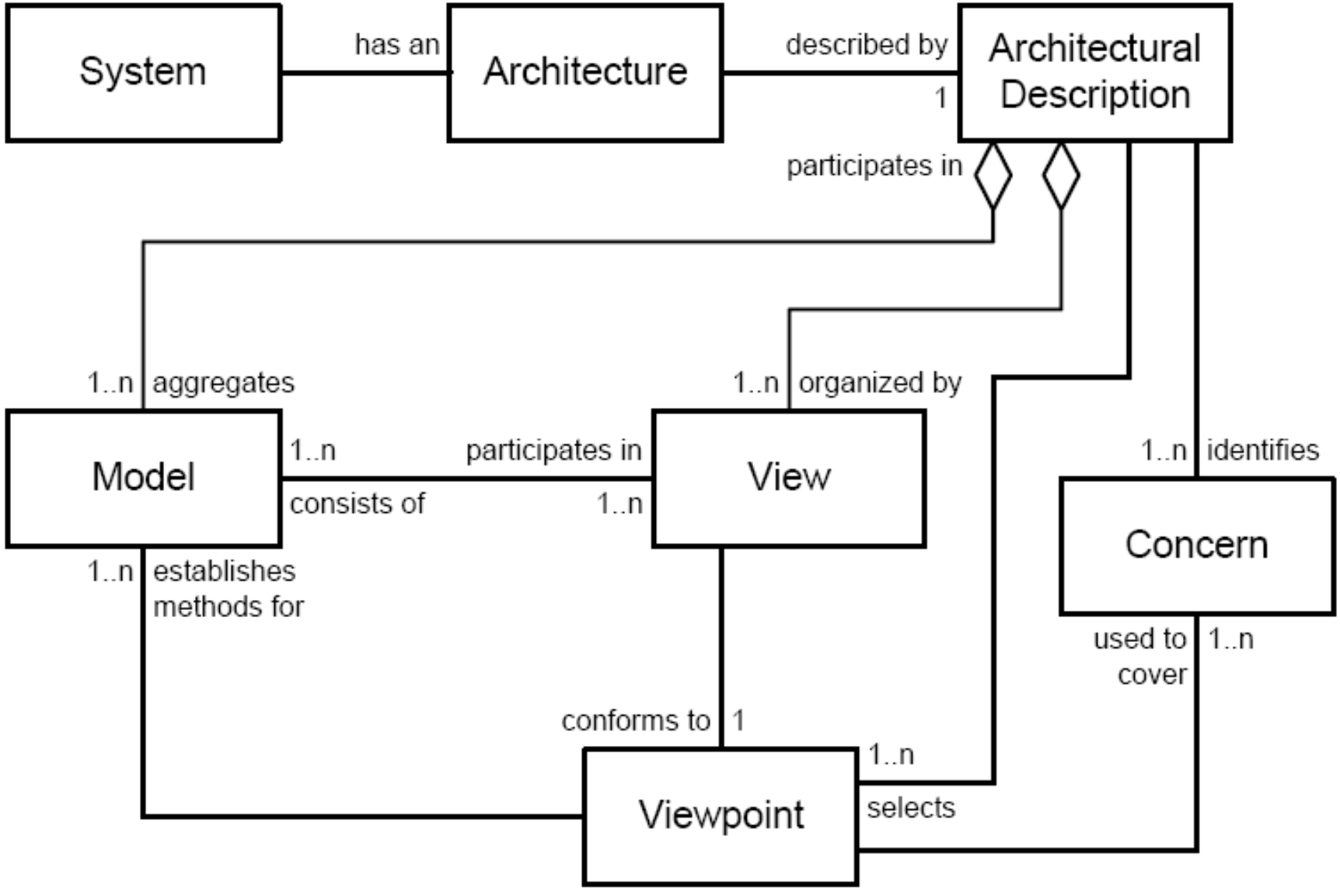
- Domänenspezifische Modelle und Wiederverwendung von Architekturmetamodelle.
  - Reduzierung der Time-to-Market ohne Budgets Erhöhung
  - Wissensvorsprung innerhalb fachlichen Domänenwissens.
- Fachlichkeit innerhalb des Software-Entwicklungszyklus wird Bedürfnis der Kunden
  - hoher Flexibilität/Agilität der unterliegenden Geschäftsprozesse
  - ohne unnötige Beschränkung durch technologische Vorgaben gerecht.

Vorgaben der OMG

- Konkreten Vorgehensanweisungen in MDA-Projekten immer noch vage.
- Eher Konzept-Charakter, als Spezifikation.
- OMG beschreibt einige Basiskonzepte[1,2].
  - Im Folgenden die Wichtigsten.
  - Vorab Begriffsbestimmungen
    - Terminologie IEEE 1471
    - Begriffe mit zentraler Rolle innerhalb MDA.

1 Object Management Group (OMG): *Model Driven Architecture – A Technical Perspective*. http://www.omg.org/cgi-bin/apps/doc?ormsc/01-07-01 (letzter Abruf Mai 2005)
2 Object Management Group (OMG): *MDA Guide Version 1.0.1*. http://www.omg.org/cgi-bin/doc?omg/03-06-01 (letzter Abruf Mai 2005).

Terminologie IEEE1471



Terminologie IEEE1471

System

- Aus Teilen zusammengesetztes und strukturiertes Ganzes.
- Hat eine Funktion und erfüllt einen Zweck.

Architecture

- Organisation der Teile eines Systems und deren Verhalten.
- Beziehungen unter den einzelnen Komponenten.
  - Beziehungen können an Bedingungen geknüpft sein.
    - Müssen zur Erreichung des Systemzwecks erfüllt werden.

Architectural Description

- Besteht aus Menge aller Artefakte,
  - die zur Beschreibung einer Architektur benötigt werden.
- Typischerweise aus Modellen des Systems und unterstützender Dokumentation.

Terminologie IEEE1471

Concern

- Spezifische Anforderungen oder Bedingungen,
  - Müssen erfüllt werden, um erfolgreiche Erfüllung des Systemzwecks zu gewährleisten.
- Im Folgenden übersetzt als Aspekt.

Model

- Beschreibt oder spezifiziert ein System zu einem bestimmten Zweck.
- Erfasst alle relevanten Aspekte.
  - z.B.: Struktur, Verhalten, Funktion und Umwelt des Systems.

Terminologie IEEE1471

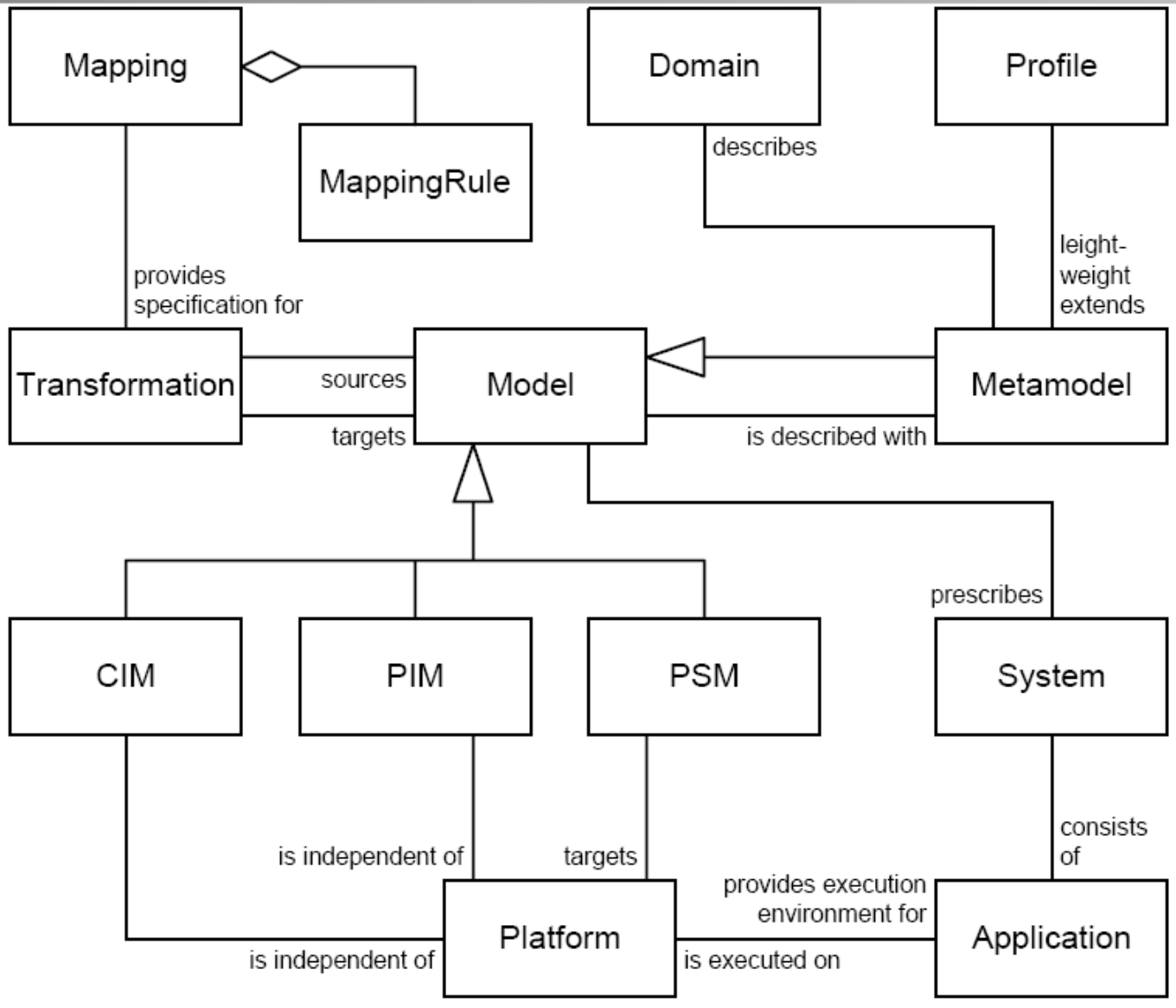
Viewpoint

- Beschreiben Elemente und Konstrukte,
  - Ermöglichen Erstellung eines Modells zur Beschreibung von Systemaspekten.
- Einfacher gesagt:
  - Regeln, Techniken bzw. Methoden sowie Notationsmittel zur Erstellung eines Modells.

View

- Kohärente Menge von Modellen,
  - Beschreibt bestimmte Aspekte eines Systems
  - Verbirgt irrelevante Aspekte.
- Wird als Sicht auf System bezeichnet.
- Wird mit Regeln und Notation des entsprechenden Viewpoints dargestellt.

Zentrale MDA-Begrifflichkeiten



Zentrale MDA-Begrifflichkeiten

Metamodel

- Menge von Elementen mit denen Modelle erstellt werden können.
- Enthält:
  - Regeln, die bei der Erstellung des Modells beachtet werden müssen (abstrakte Syntax).
  - Bedeutung der Elemente und Elementkonstellationen (Semantik).

Profile

- Leichtgewichtige Erweiterung eines Metamodells.
- Stellt neue Modellelemente zur Verfügung oder redefiniert bzw. erweitert die Semantik oder Syntax bereits bestehender Elemente.
- Können Bedingungen, die an Elemente der Modelle gestellt werden, nur verschärfen.
  - Müssen sich den Regeln eines Design-by-Contracts [DBC] unterordnen.

Zentrale MDA-Begrifflichkeiten

Domain

- Abgrenzbares, kohärentes (Wissens-)Gebiet.
- Im Metamodell (respektive Profil) werden Konzepte einer Domäne in Bezug gesetzt und beschrieben.
- Zusammen mit ihrem sprachdefinierenden Charakter bezeichnet man Metamodell und Profil daher als Domänenspezifische Sprache (engl. Domain-Specfic Language, DSL).

Application

- Zu erstellendes Stück Software, umfasst zu entwickelnde Funktionalität.
- System kann sich aus einer oder mehreren Anwendung(en) zusammensetzen, die auf einer oder mehreren Plattform(en) ausgeführt werden.

Zentrale MDA-Begrifflichkeiten

Plattform

- Ausführungsumgebung für Anwendung.
- Zugriff auf Funktionalität über Schnittstellen.
  - Kenntnis über Implementierung nicht nötig.
- Bsp.: Plattformen im Bereich Betriebssysteme (Unix, Windows, OS/390).
- Bsp.: Programmiersprachen (C++, Java, C#).
- Bsp.: Middleware (CORBA, Java EE, .NET).
- Setzen of in Art eines Plattform-Stacks aufeinander auf.
- Hardware eines Computers bildet Plattform für das Betriebssystem.
- Betriebssystem ist Plattform für die einzelnen Anwendungen usw.

Zentrale MDA-Begrifflichkeiten

Computation Independent Model (CIM)

- Liefert eine Sicht auf Gesamtsystem unabhängig davon von Implementierung.
- Modell im Vokabular seiner Domäne beschrieben und betont Anforderungen an das System und seine Umwelt.
- Synonyme: Business Model, Domain Model

Platform Independent Model (PIM)

- Beschreibt formale Struktur und Funktionalität eines Systems.
- Ist unabhängig implementierungstechnischen Details
  - Abstrahiert von zugrunde liegenden Plattform.

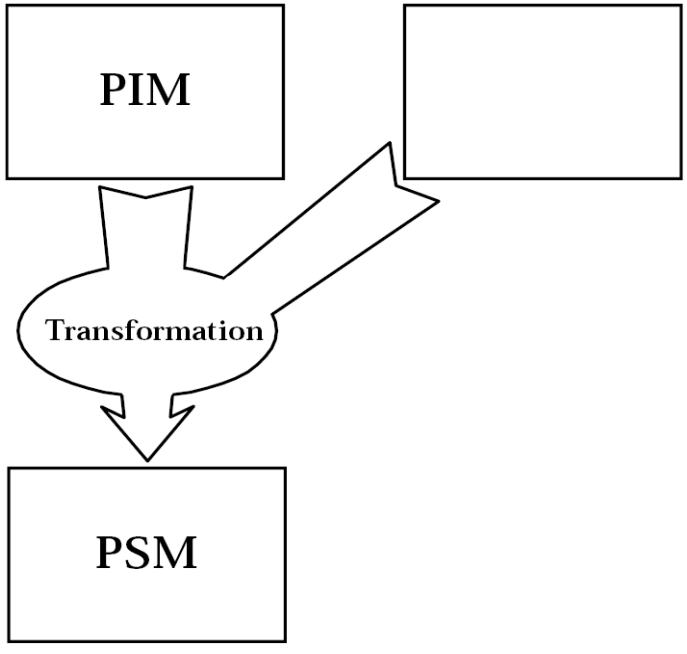
Platform Specific Model (PSM)

- PIM mit plattform-abhängigen Informationen.
- Wird zur Implementierung, falls es alle Informationen zur Konstruktion und Betrieb eines Systems liefert.
  - Modelle können auch direkt ausgeführt werden (executable models).

(Modell-)Transformation

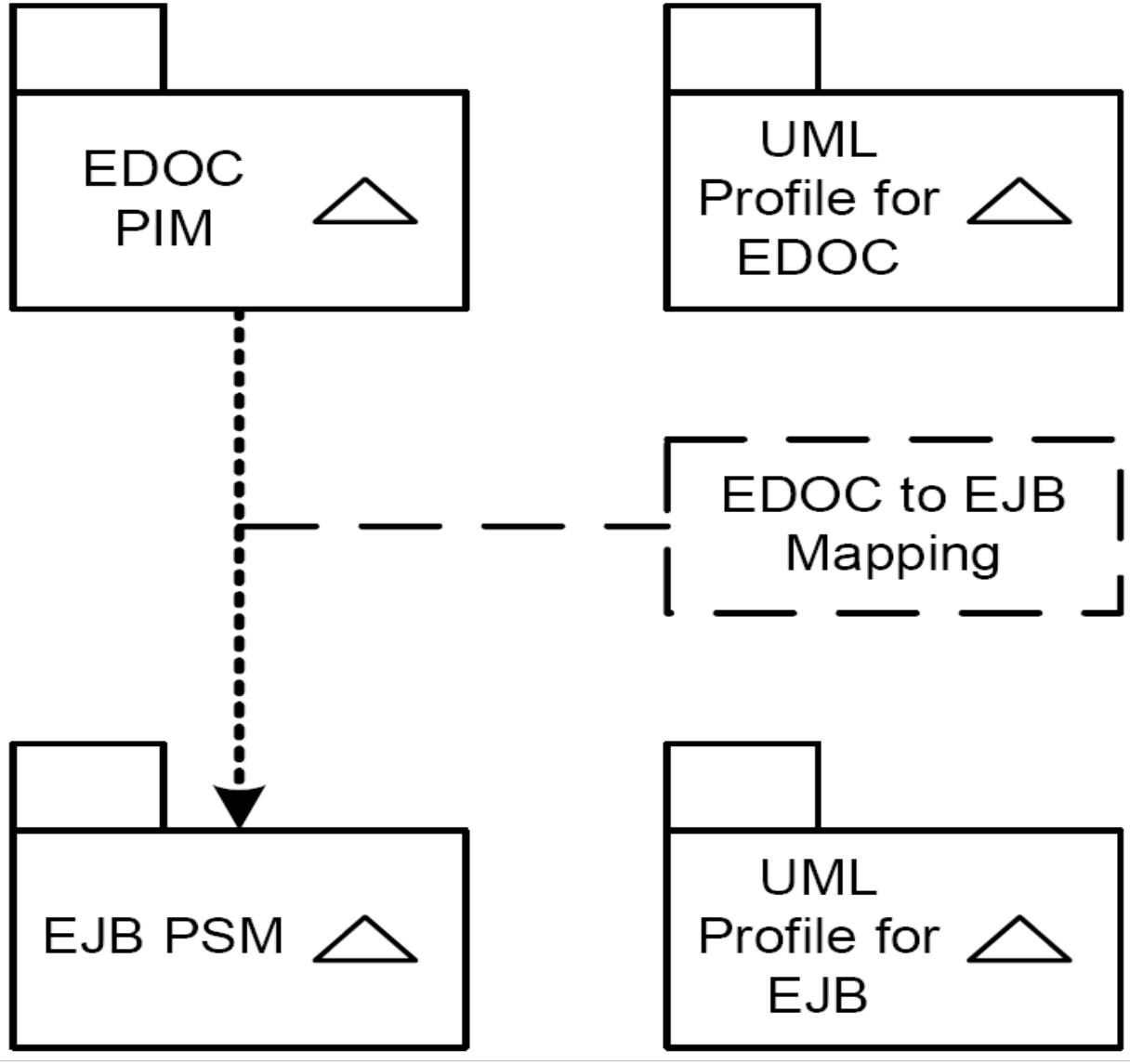
(Modell-)Transformation

- Prozess der Umwandlung eines Modells in ein anderes Modell des
- gleichen Systems.
- Spezifikation für Transformation in Mapping Rules festgehalten,
- Mapping Rules in Mapping gebündelt sind.
- Dieses Muster ist das Kernstück des MDA-Ansatzes.
- Zur Transformation in ein ein PSM werden das PIM und „weitere Informationen“ benötigt (leerer Kasten).



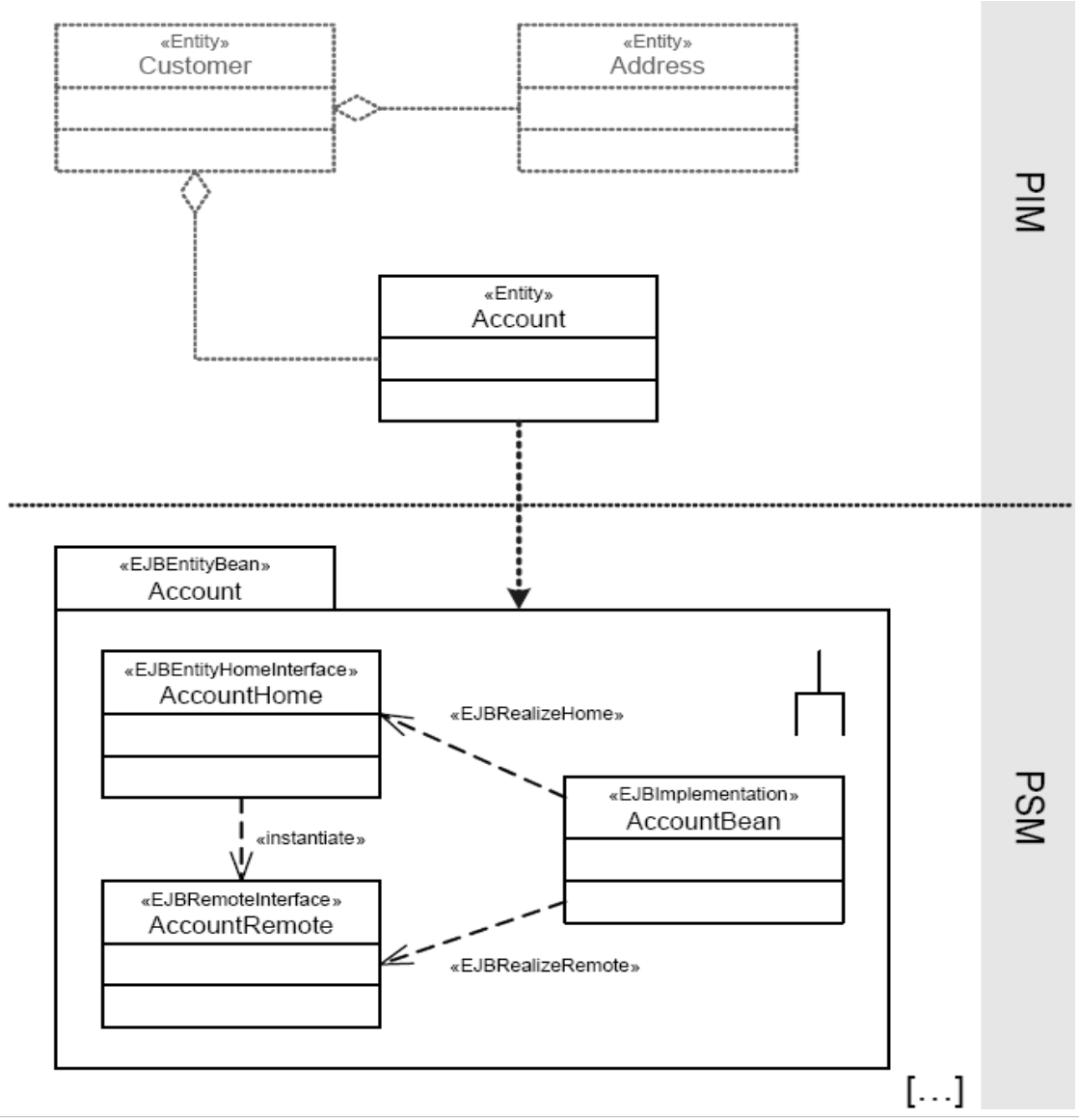
Object Management Group (OMG): *MDA Guide Version 1.0.1*. Figure2-2 http://www.omg.org/cgi-bin/doc?omg/03-06-01 (letzter Abruf März 2013).

Transformation Beispiel



Transformation Beispiel PIM → PSM

- PIM auf höherer Abstraktionsebene angesiedelt.
- «Entity» der Klasse Account trägt Semantik.
  - Führt zur Abbildung dieser Klasse in EntityBean des PSM.



Gruhn, Pieper, Röttgers: MDA - Effektives Software-Engineering mit UML 2 und Eclipse, Springer-Verlag Berlin Heidelberg 2006 Abb. 2.09

Transformation Beispiel PSM → Code

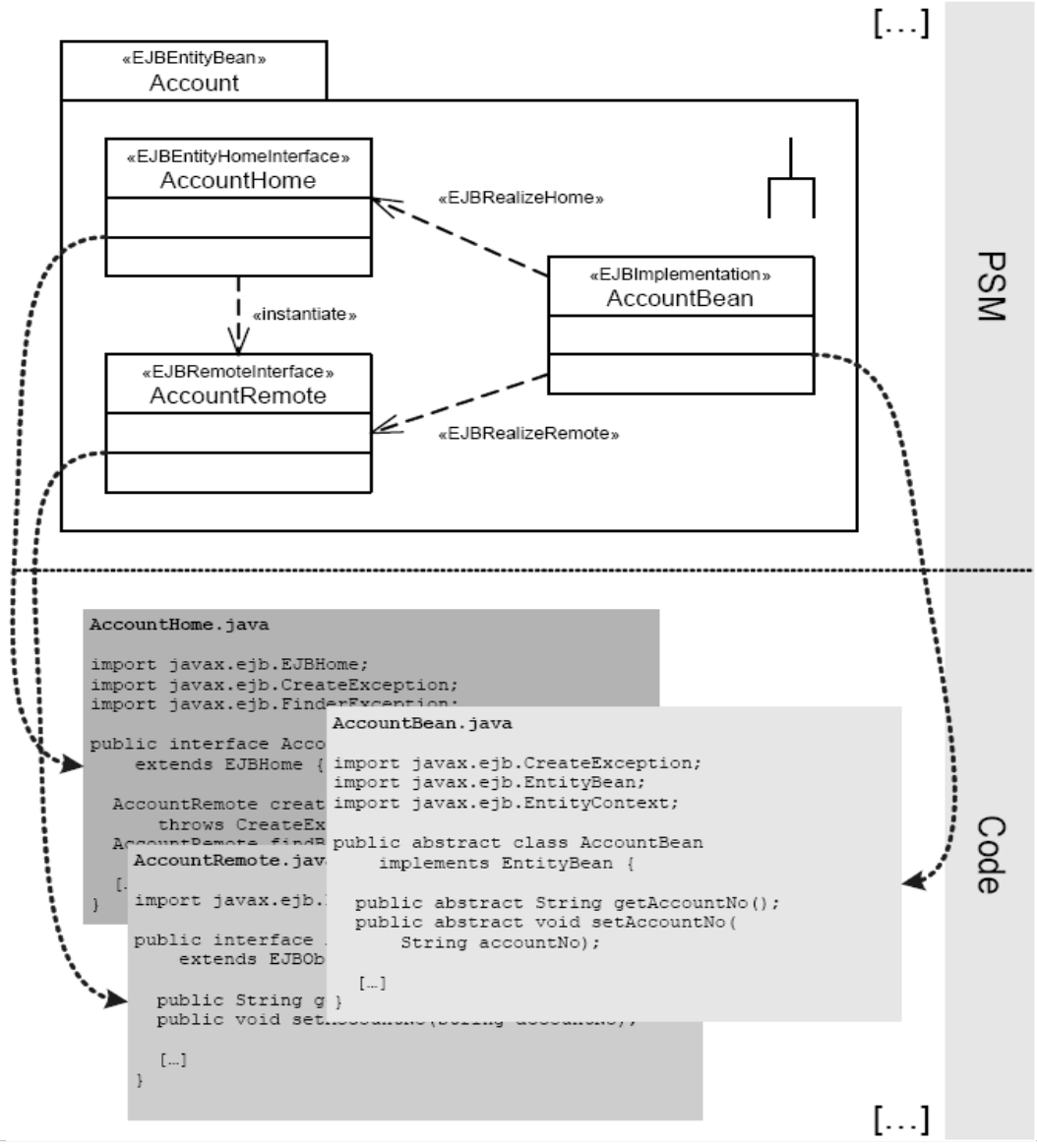
Expandiert in BeanClass sowie RemoteInterface und HomeInterface

- Wie Komponentenmodell (EJB-Spezifikation) verlangt.

Der Übersichtlichkeit wegen nicht dargestellt:

- Operationen zum EJB-Lebenszyklus (ejbCreate, ejbActivate, …).
- Deployment-Deskriptoren (ejb-jar.xml, ..., sowie hersteller-spezifische Deskriptoren).
- SQL-Skripte (z.B.: zum Aufsetzen des Datenbankschemas).
- Testskripte und ähnliches

werden jedoch generiert.



Gruhn, Pieper, Röttgers: MDA - Effektives Software-Engineering mit UML 2 und Eclipse, Springer-Verlag Berlin Heidelberg 2006 Abb. 2.10

Standards im Dunstkreis der MDA

Meta Object Facility (MOF)

- Modellbasierte Sprache der OMG zur Definition von Metamodellen.
- Bsp.: sämtliche Standards des UML-2.x-Stacks mit ihr beschrieben.

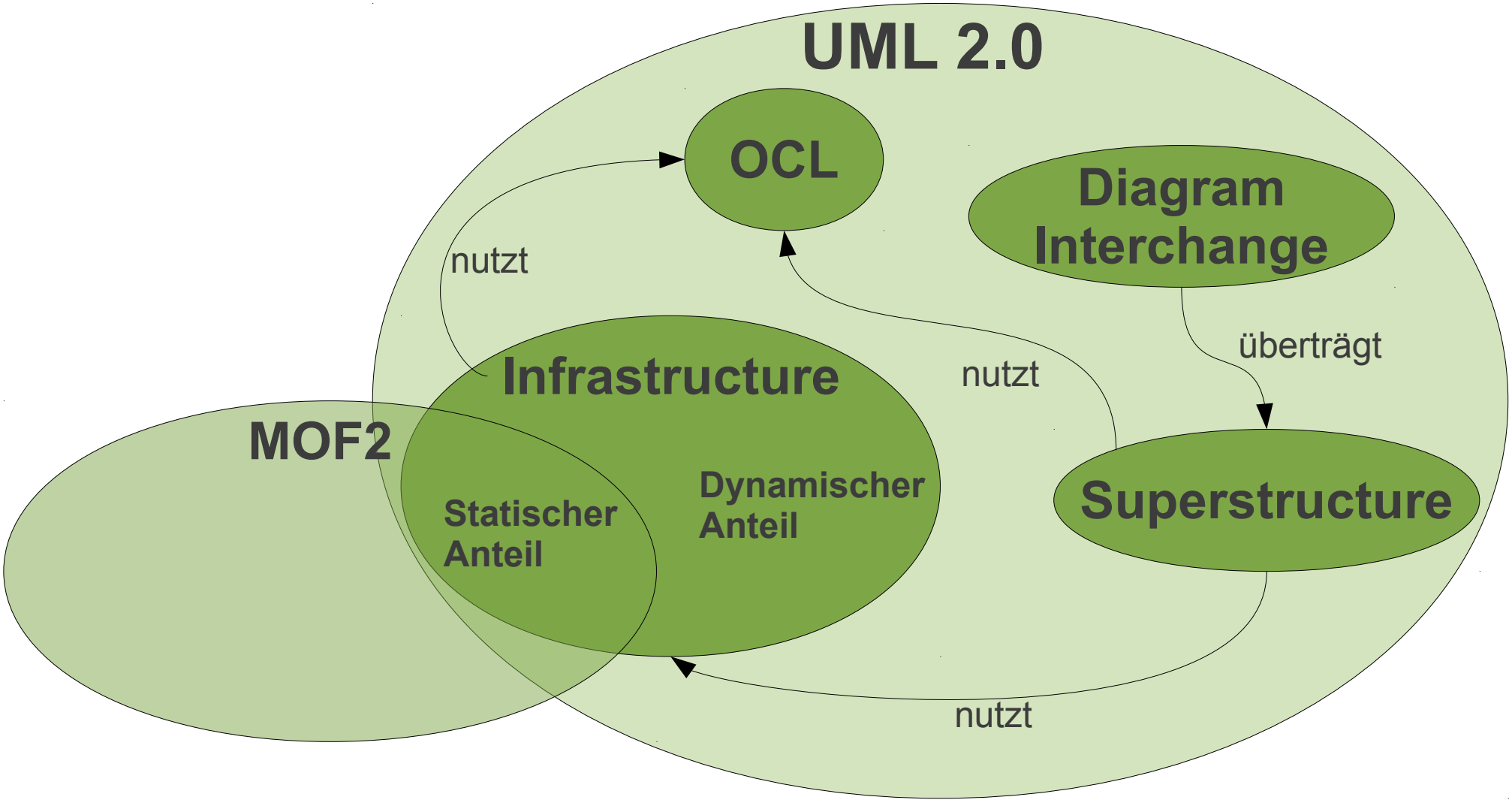
Unified Modeling Language (UML)

- Mittel der Wahl zur Erstellung der Modelle innerhalb der MDA.

XML Metadata Interchange (XMI)

- Definiert Abbildung der MOF auf XML.
- Ermöglicht den standardisierten Austausch von beliebigen (Meta-)Modellen zwischen Tools.
  - z.B.: Transformatoren,Modellierungswerkzeugen, Codegeneratoren usw.
  - Eine Grundvoraussetzungen zum Aufbau funktionierenden MDA-Infrastruktur.

UML Spezifikation



UML Spezifikation

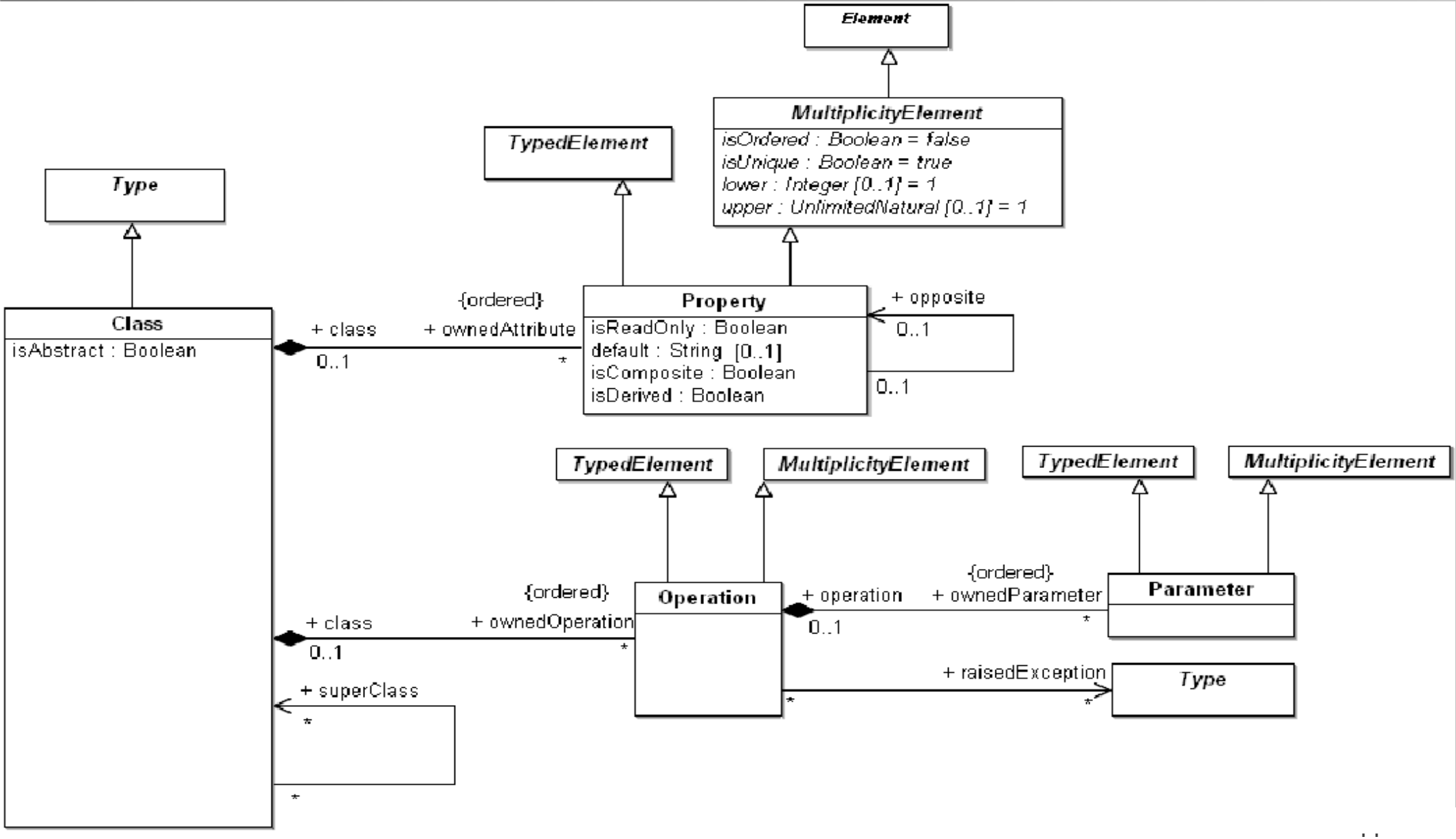
Definition der Modellierungssprache

Bestandteile

- Abstrakte Syntax
  - Modellelemente
- Statische Semantik
  - Wohlgeformtheit
- Konkrete Syntax
  - Notation, ggf. graphische Symbole
- Dynamische Semantik
  - Verwendung, Bedeutung

Metamodell
+ Constraints (OCL)

UML Metamodell Beispiel
Core::Basic:ClassDiagram



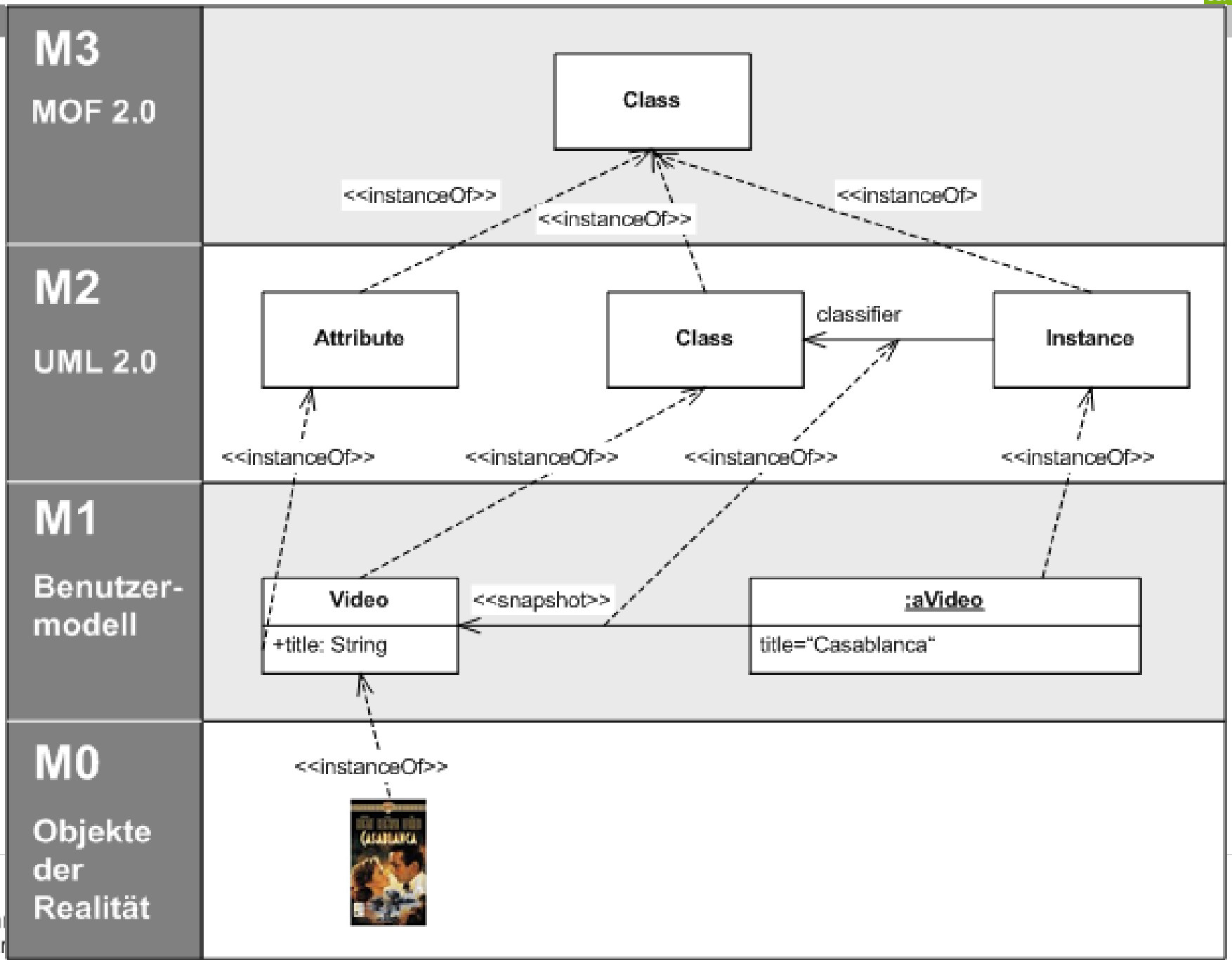
Metamodellierung

- Modelle sind Instanzen ihrer Metamodelle
- Beispiel Diagramm vs. Metamodell-Instanzen
- Metamodelle definieren Modellelemente
  - „Sprachdefinition“
- Metamodelle für:
  - Maschinenlesbarkeit
  - Validierung
  - Speicherung von Modellen (Repositories)
  - Datenaustausch/Interoperabilität
  - Definition von Transformationen

Meta Ebenen

- M0-Ebene
  - Konkret. Ausgeprägte Daten.
- M1-Ebene
  - Modelle. Zum Beispiel physikalische oder logische Daten- oder Prozessmodelle oder konkrete Ausprägungen von UML- bzw. Objekt-Modellen, welche die Daten der M0-Ebene definieren.
- M2-Ebene
  - Meta-Modelle. Definieren, wie die Modelle aufgebaut und strukturiert sind. Zum Beispiel definieren Sprachelemente wie Klassen, Assoziationen und Attribute der UML 2.0, wie konkrete UML-Modelle aufgebaut sein können.
- M3-Ebene
  - Meta-Meta-Modelle (bzw. MOF-Ebene). Abstrakte Ebene, die zur Definition der M2-Ebene herangezogen wird. Die Definition der M3-Ebene erfolgt mit den Mitteln der M3-Ebene selbst, dies stellt den Abschluss einer sonst unendlichen Metaisierung dar.

Meta Ebenen


M3
MOF 2.0
Class
<<instanceOf>>
<<instanceOf>>
<<instanceOf>
M2
UML 2.0
Attribute
Class
classifier
Instance
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
M1
Benutzer-
modell
Video
+title: String
<<snapshot>>
:aVideo
title="Casablanca"
M0
Objekte
der
Realität
<<instanceOf>>

Metamodellierung

Wege zum „eigenen“ Metamodell

- Eigenes Metamodell „from scratch“
  - aufwändig
- Direkte, beliebige Erweiterung/Anpassung eines Metamodells
  - Setzt uneingeschränkten Zugriff auf das Metamodell voraus und verletzt ggf. exisitierende Semantik
- Erweiterung des UML-Metamodells durch
  - UML Profile
  - Vorgesehene Schnittstelle zum UML-Metamodell
  - Lediglich Verfeinerung

UML Profil
Erweiterungsmöglichkeiten

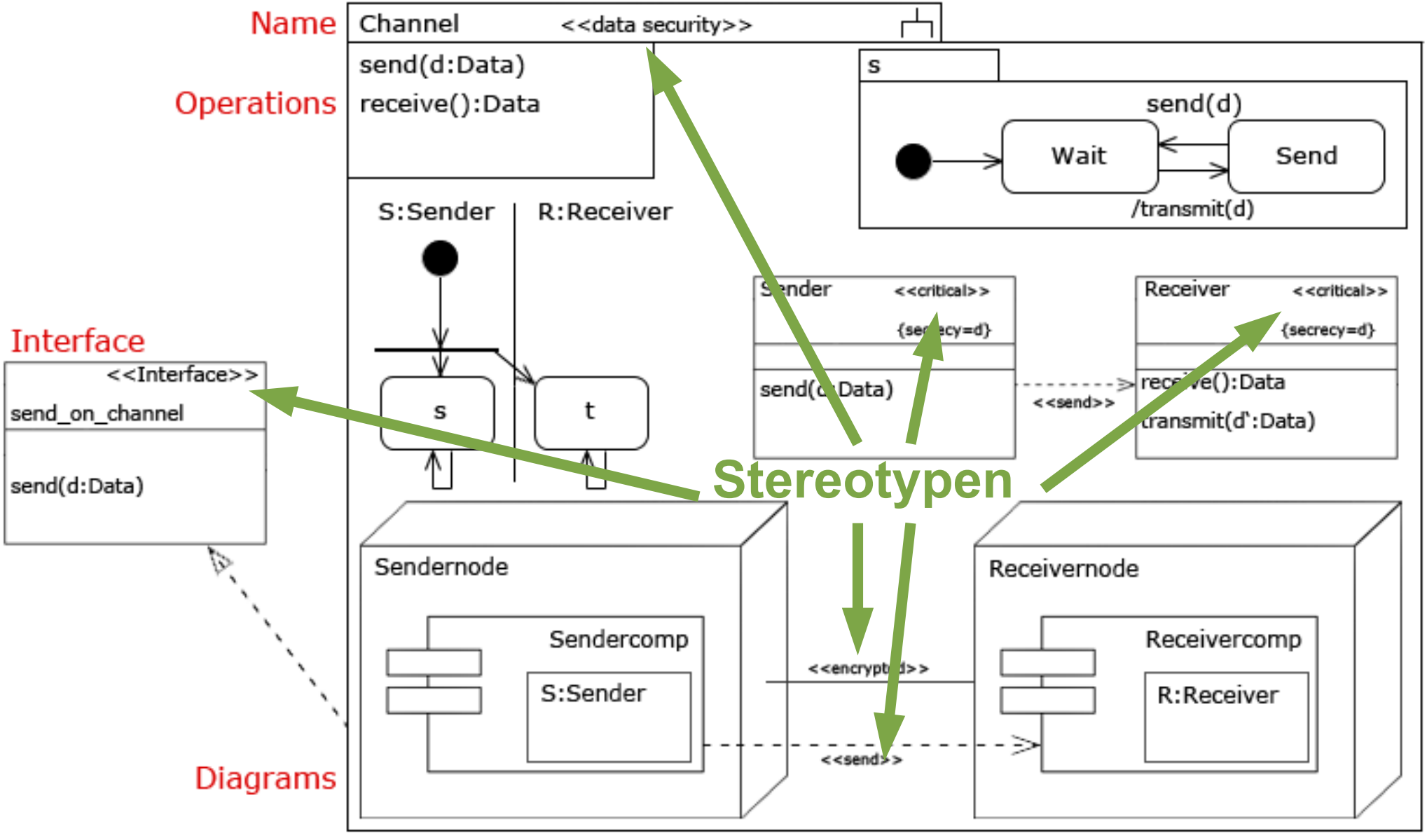
Stereotypen: spezialisiert die Benutzung von Modellelementen <<label>>.

Tagged value: fügt {tag=value} Paare zu stereotypisierten Elementen hinzu.

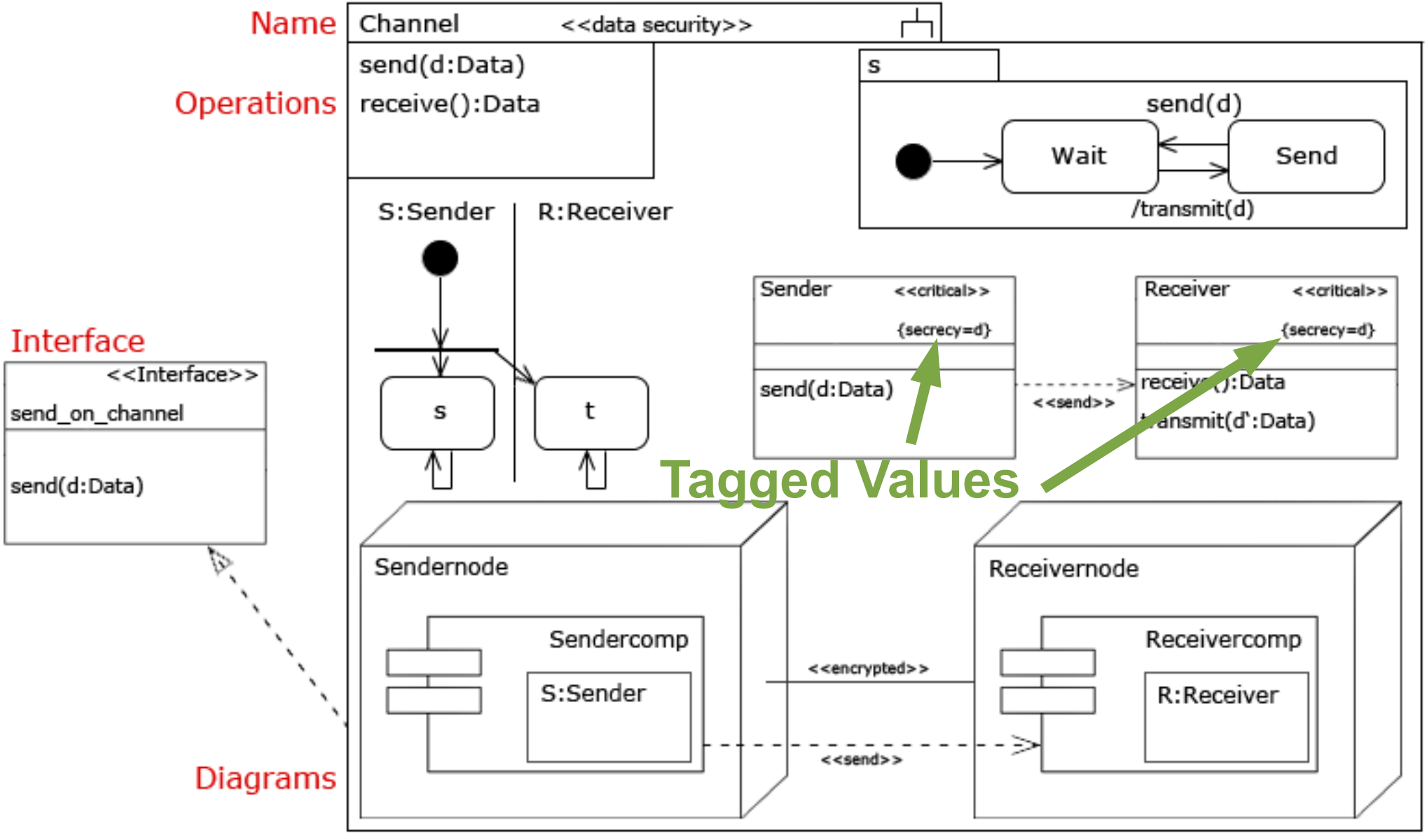
Constraint: verfeinert die Semantik eines stereotypisierten Elements.

Profil: sammelt obige Informationen.

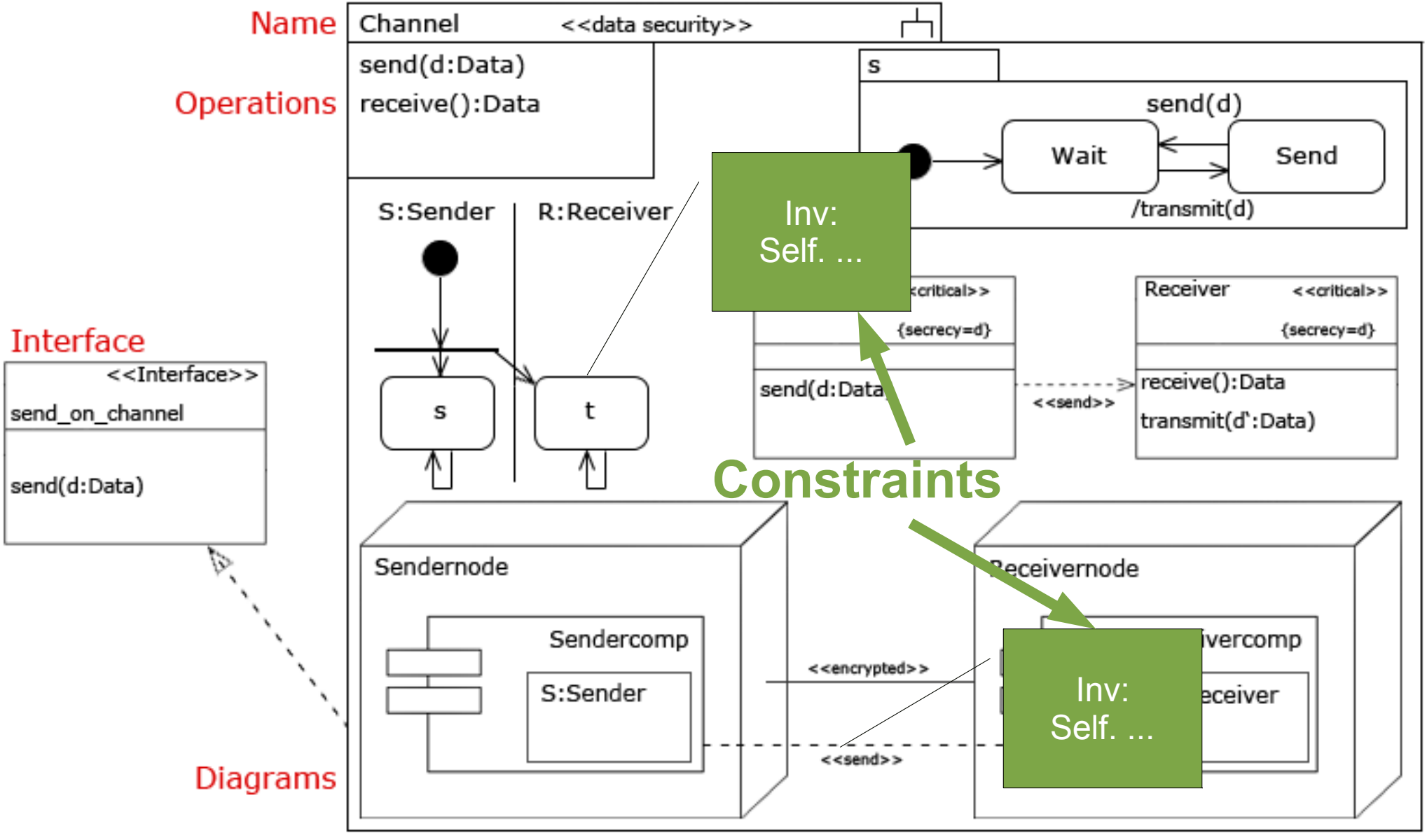
UML Profil
Erweiterungsmöglichkeiten



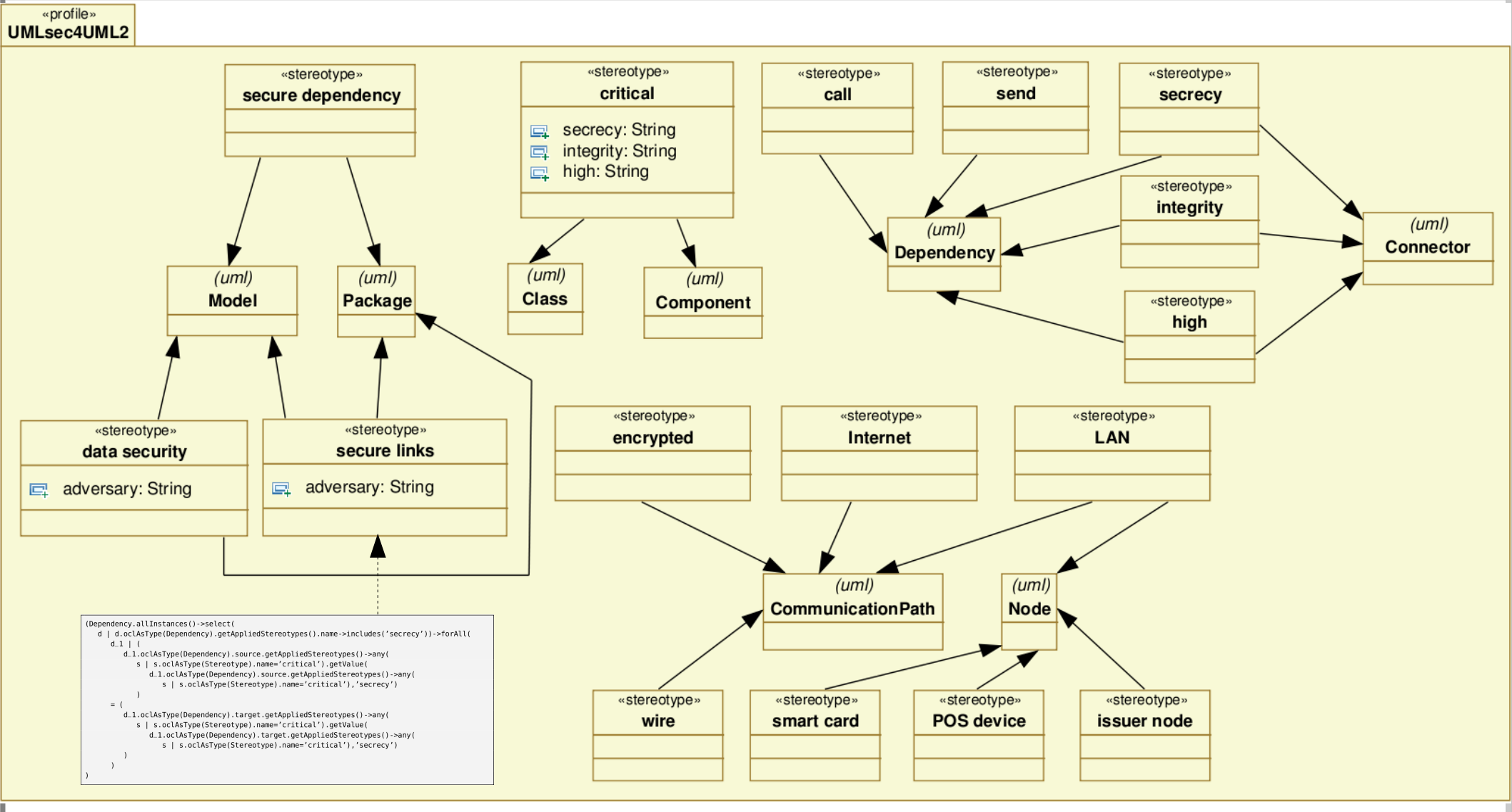
UML Profil
Erweiterungsmöglichkeiten



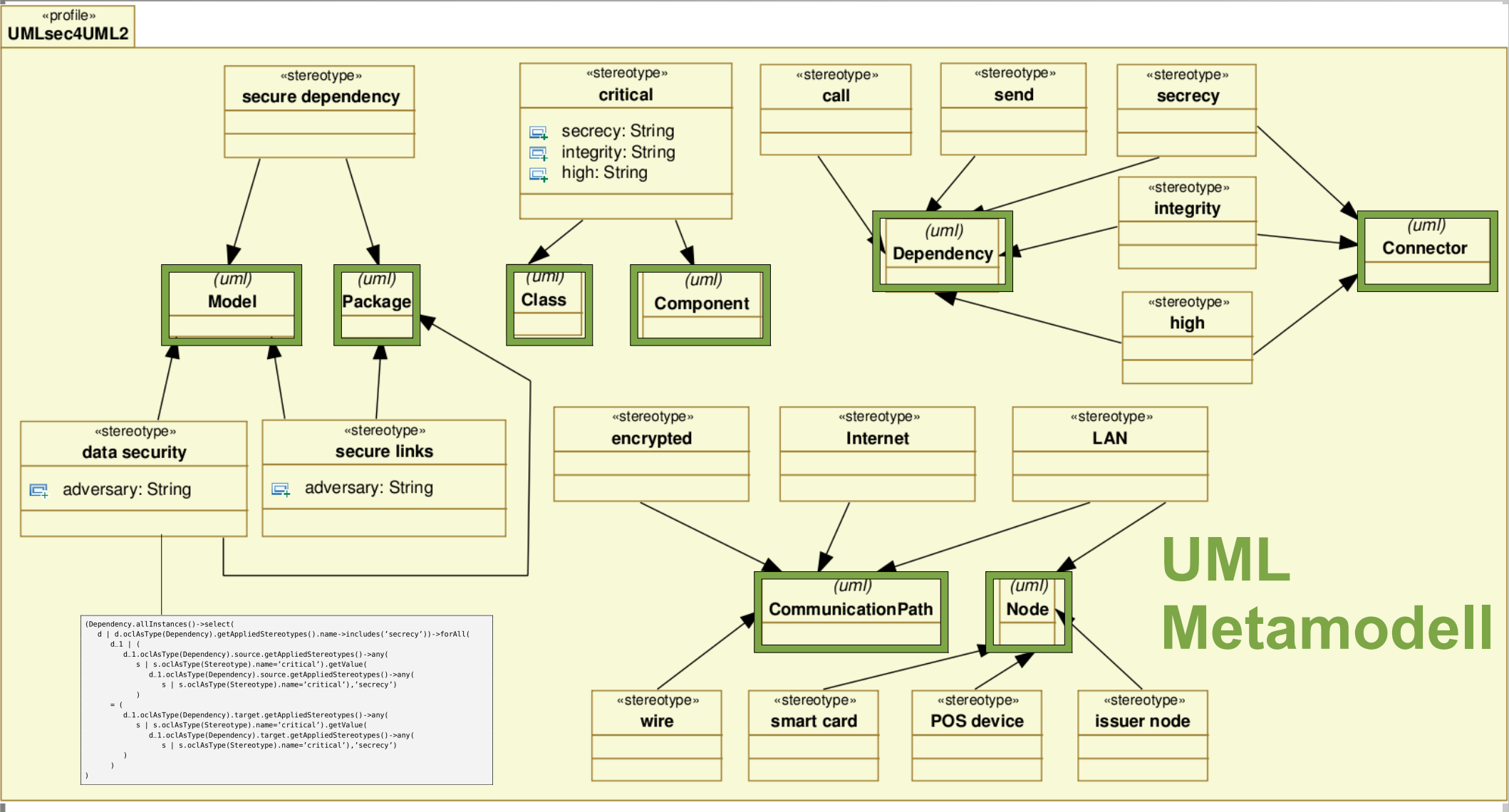
UML Profil
Erweiterungsmöglichkeiten



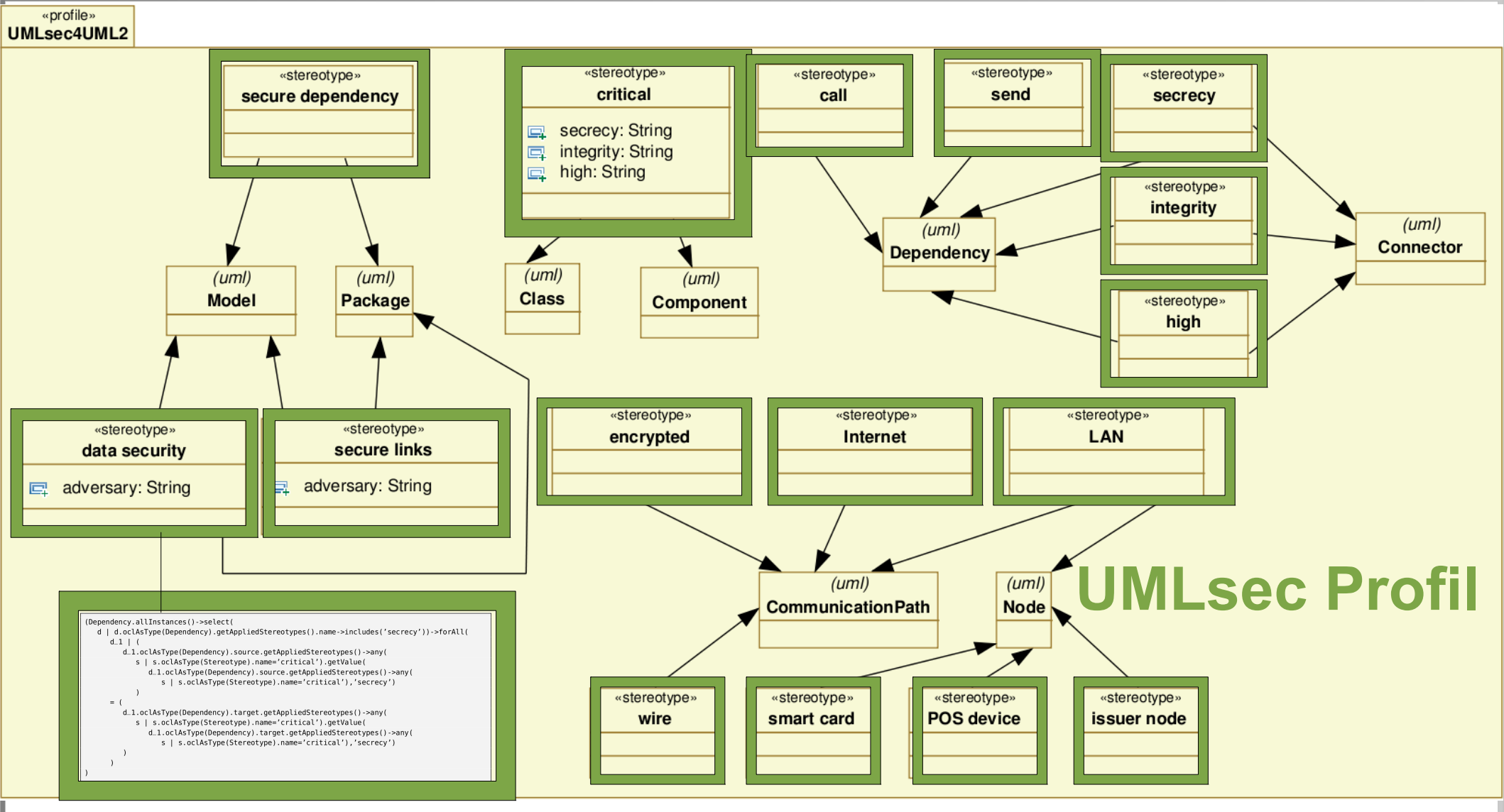
UML Profil
Beispiel



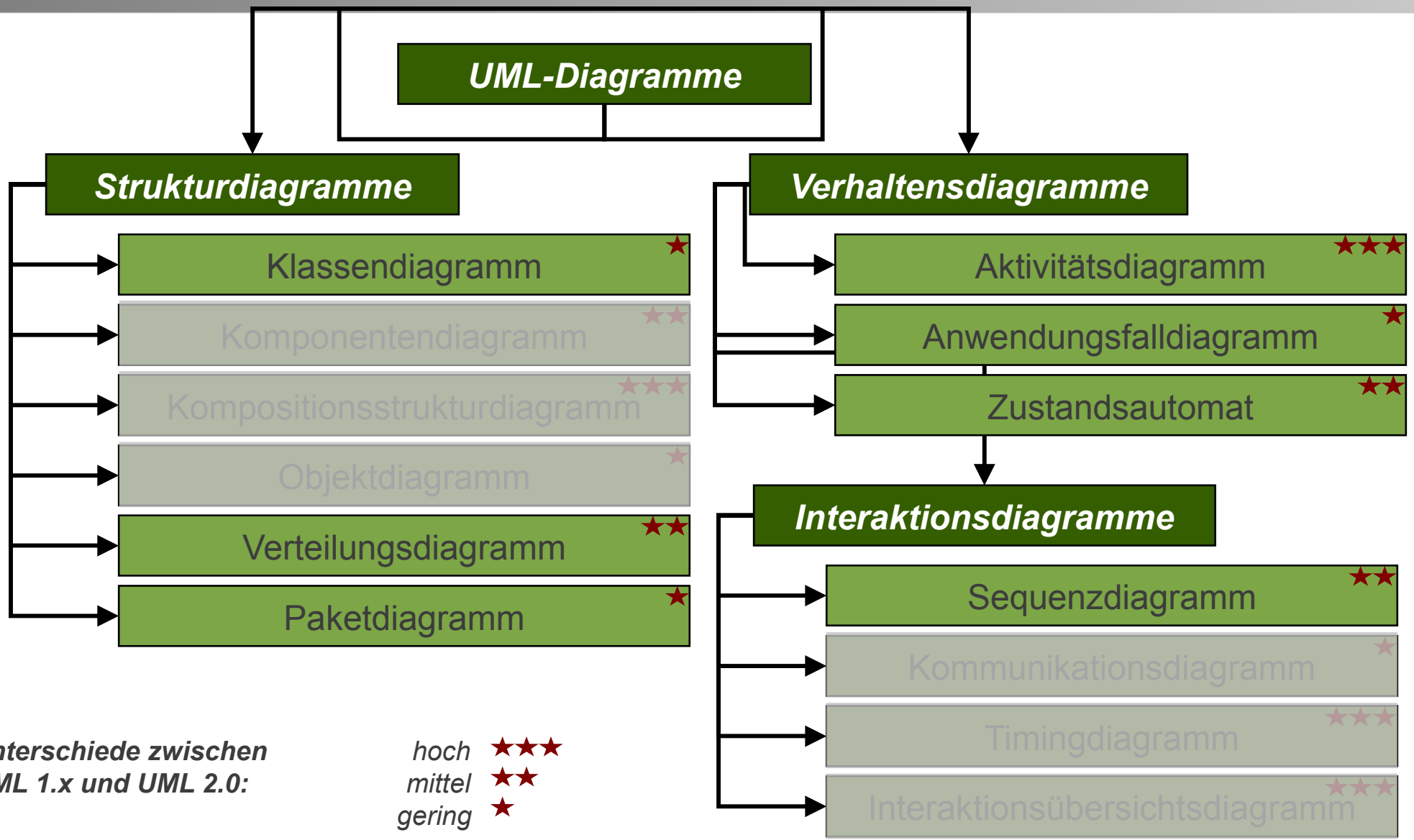
UML Profil
Beispiel



UML Profil
Beispiel



UMLsec-relevante Diagrammtypen



Unterschiede zwischen UML 1.x und UML 2.0:

hoch ★★★
mittel ★★
gering ★

UMLsec-relevante Diagrammtypen

Anwendungsfalldiagramm: Enthält die Anforderungen an das System

Klassendiagramm: Datenstruktur des Systems

Zustandsdiagramm: dynamisches Verhalten der Komponenten

Aktivitätsdiagramm: Steuerung des Ablaufs zwischen den Komponenten

Sequenzdiagramm: Interaktion durch Nachrichtenaustausch

Verteilungsdiagramm: Physikalische Umgebung

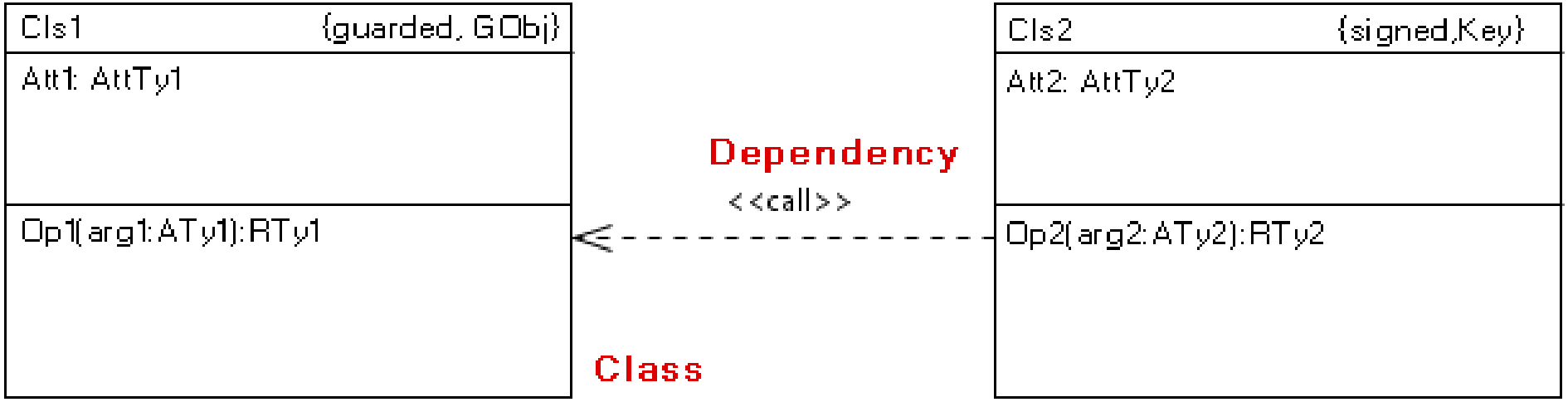
Paket/Subsystem: Fasst Diagramme eines Systemteils zusammen

UMLsec Anmerkungen

- UML2 Vorstellung auf UMLsec relevante Diagramme beschränkt
- UMLsec auf UML 1.5 definiert
  - Beispiele können von UML 2 Abweichen
- UMLsec wurde zum Großteil schon auf UML 2 angehoben (Siehe UMLSec4UML2 Profil), aber um Verwirrung vorzubeugen alle Beispiele in UML 1.5

Klassenstruktur des Systems.

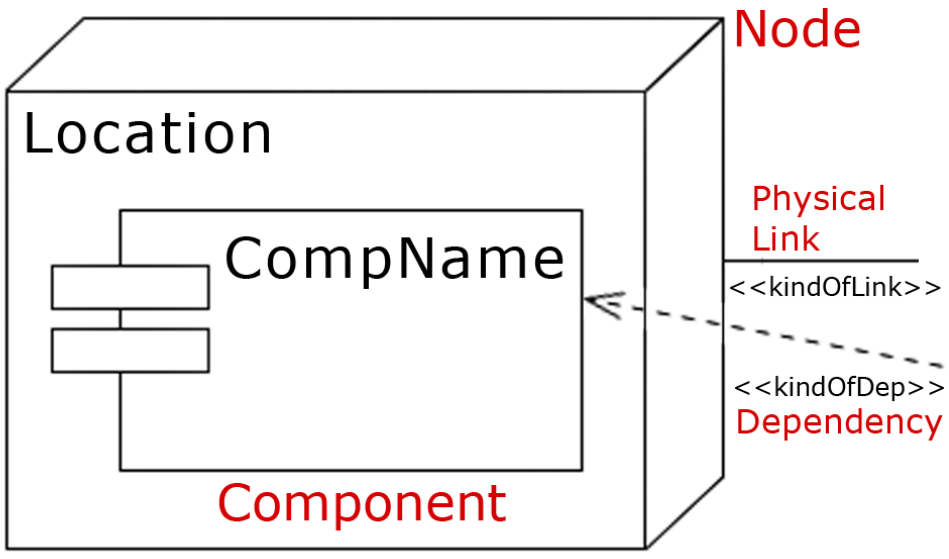
Klasse mit Attributen und Operation / Signalen,
Beziehungen zwischen Klassen.



UML im Schnelldurchlauf
Verteilungsdiagramm

- Problem:
  - Es soll die Verteilung der Software des Systems auf die Hardware beschrieben werden.
- Diese zentrale Frage beantwortet das Diagramm:
  - Wie sieht das Einsatzumfeld (Hardware, Server, Datenbanken etc.) des Systems aus?
  - Wie werden die Komponenten zur Laufzeit wohin verteilt?

UMLsec Beispiel Verteilungsdiagramm

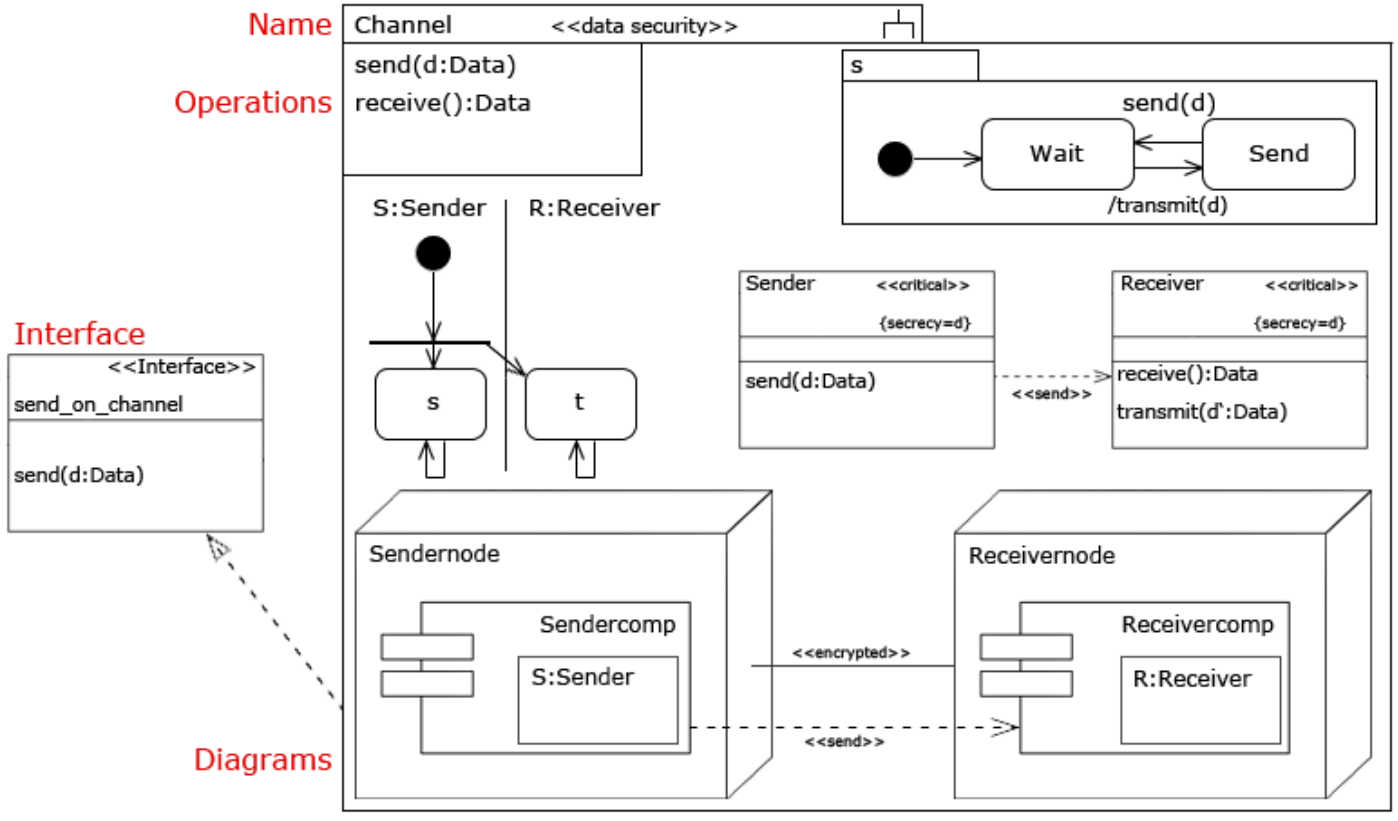


Erklärt die physikalische Ebene, auf der das System implementiert wird.

UML im Schnelldurchlauf
Paketdiagramm

- Problem:
  - Große Softwaresysteme mit mehreren 100 Klassen lassen sich nicht mehr von einer Person überblicken.
- Diese zentrale Frage beantwortet das Diagramm:
  - Wie kann ich mein Modell so schneiden, dass ich den Überblick bewahre?
- Diese Stärken hat das Diagramm:
  - organisiert das Systemmodell in größeren Einheiten durch logische Zusammenfassung von Modellelementen
  - Modellierung von Abhängigkeiten und Inklusionen ist möglich

UMLsec Beispiel
Paketdiagramm



Kann benutzt werden, um UML Element zu einer Gruppe zusammen zu fügen.

UML im Schnelldurchlauf
Aktivitätsdiagramm

- Problem:
  - Es sollen Abläufe, z.B. Geschäftsprozesse, modelliert werden. Im Vordergrund steht dabei eine Aufgabe, die in Einzelschritte zerlegt werden soll.
  - Es sollen Details eines Anwendungsfalles festgelegt werden.
- Diese zentrale Frage beantwortet das Diagramm:
  - Wie realisiert mein System ein bestimmtes Verhalten?

UML im Schnelldurchlauf
Aktivitätsdiagramm

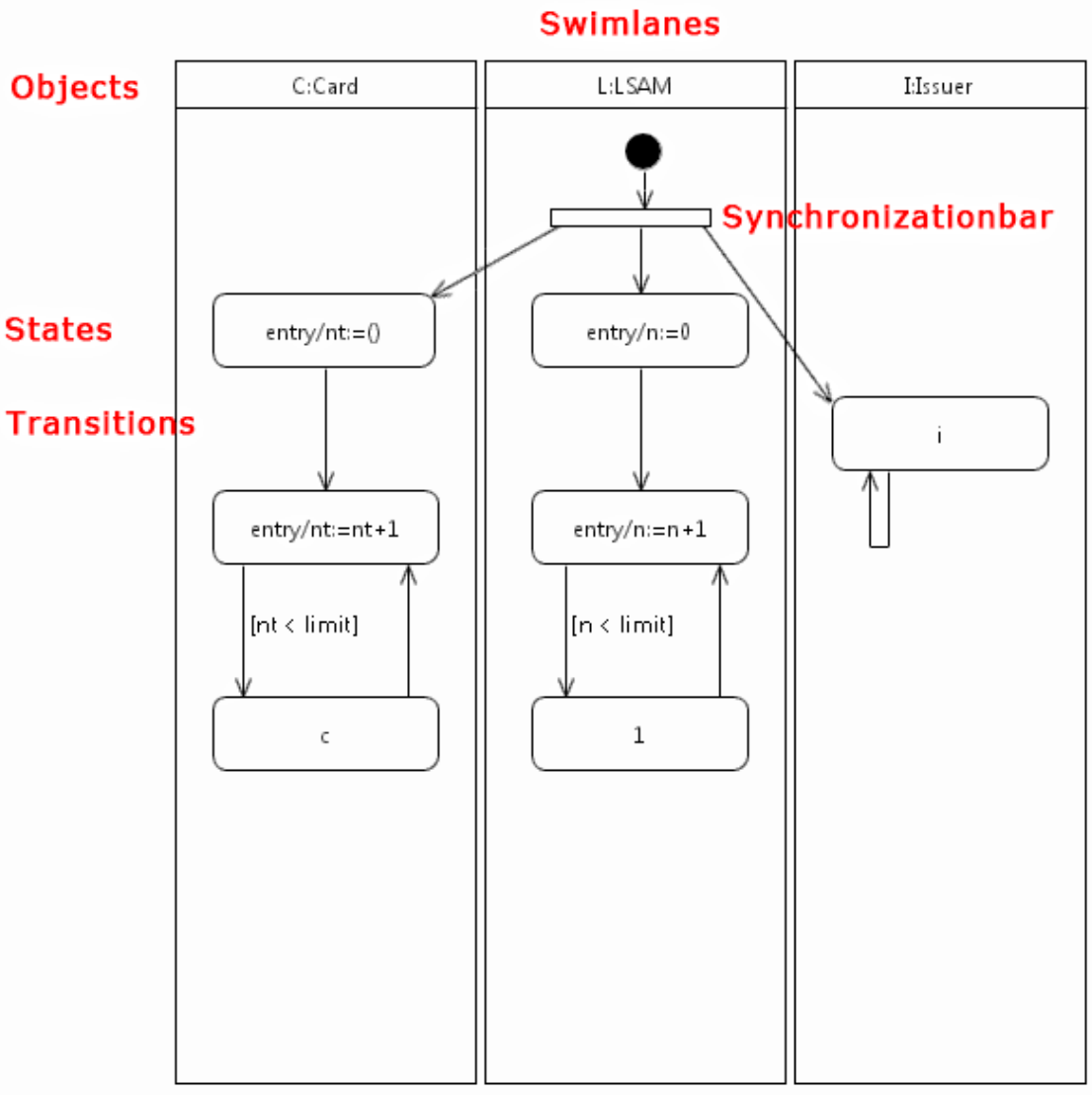
- Änderungen gegenüber früheren UML-Versionen
  - sind jetzt keine Sonderform der Zustandsdiagramme mehr, sondern basieren auf erweiterten Petri-Netzen
  - damit sind viele Einschränkungen beseitigt:
    - verbesserte Testbarkeit
    - fast automatisch erkennbare Verklemmungsfreiheit
    - bessere Unterstützung paralleler Flüsse
    - Ausführbarkeit fast vollständig möglich
    - mehr Flexibilität in der Modellierung durch das Tokenkonzept

UML im Schnelldurchlauf
Aktivitätsdiagramm stark geändert in UML2

- Tokenkonzept
  - Ein Token (auch: Marke) zeigt an, an welchem Punkt sich der Ablauf gerade befindet.
  - Es können beliebig viele Token unterwegs sein (parallele Abläufe).
  - Token werden graphisch nicht dargestellt, sondern dienen nur der Erklärung der Abläufe.

UMLsec Beispiel Aktivitätsdiagramm

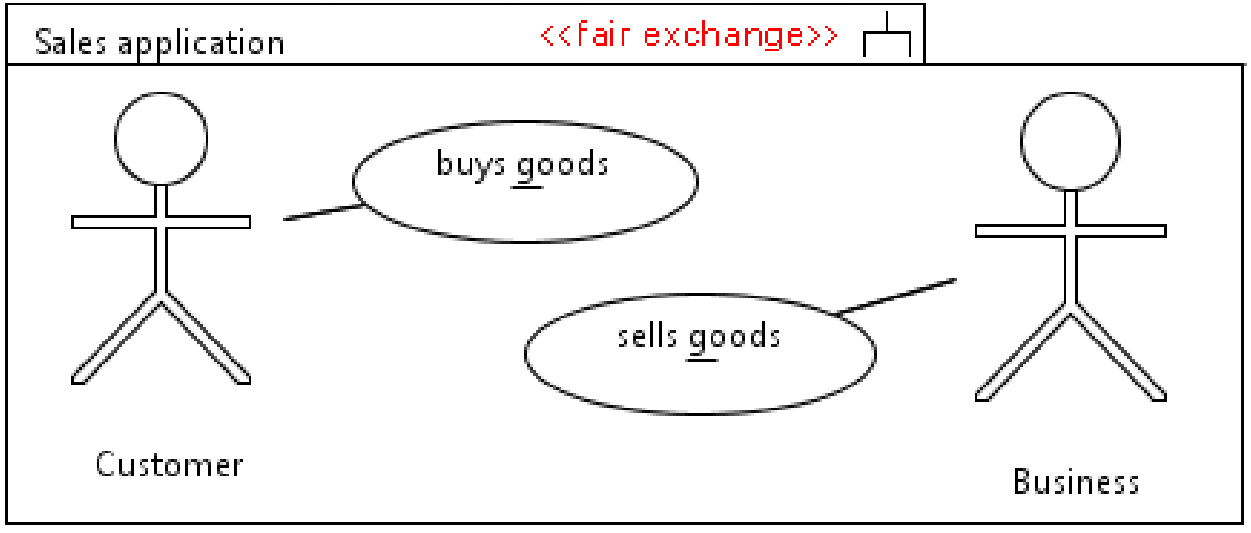
Spezifiziert den Kontrollfluss zwischen Komponenten desselben Systems. Ist auf einer höheren Abstraktionsebene als Zustands- und Sequenzdiagramme.



UML im Schnelldurchlauf
Anwendungsfalldiagramm

- Problem:
  - Das externe Verhalten des Systems soll aus Nutzersicht dargestellt werden.
- Diese zentrale Frage beantwortet das Diagramm:
  - Was leistet mein System für seine Umwelt (Nachbarsysteme, Stakeholder)?
- Diese Stärken hat das Diagramm:
  - präsentiert die Außensicht auf das System
  - geeignet zur Kontextabgrenzung
  - hohes Abstraktionsniveau, einfache Notationsmittel

UMLsec Beipspiel
Anwendungsfalldiagramm



Spezifiziert einen Anwendungsfall des Systems:
Szenario einer Funktionalität, die einem Benutzer oder einem anderen System angeboten wird.

Akteure die mit einer Aktivität verknüpft sind.

UML im Schnelldurchlauf
Zustandsautomat

- Beschreibung des Verhaltens von Anwendungsfällen
  - Die formale Modellierung von Anwendungsfällen hat folgende Vorteile:
    - Sie sind eindeutig und weniger interpretierbar.
    - Es können Testfälle abgeleitet werden.
- Beschreibung des Verhaltens von Klassen
  - Dem Attribut einer Klasse wird im Allgemeinen ein Datentyp zugeordnet.
  - Wenn der Datentyp eine endliche Menge von gültigen Werten besitzt, dann ergeben sich die verschiedenen Zustände der Klasse aus allen möglichen Kombinationen dieser Zustandsvariablen.

UMLsec Beispiel
Zustandsautomat

Dynamisches Verhalten der einzelnen Komponenten.

Die Eingabe verursacht einen Zustandsübergang und möglicherweise eine Ausgabe.

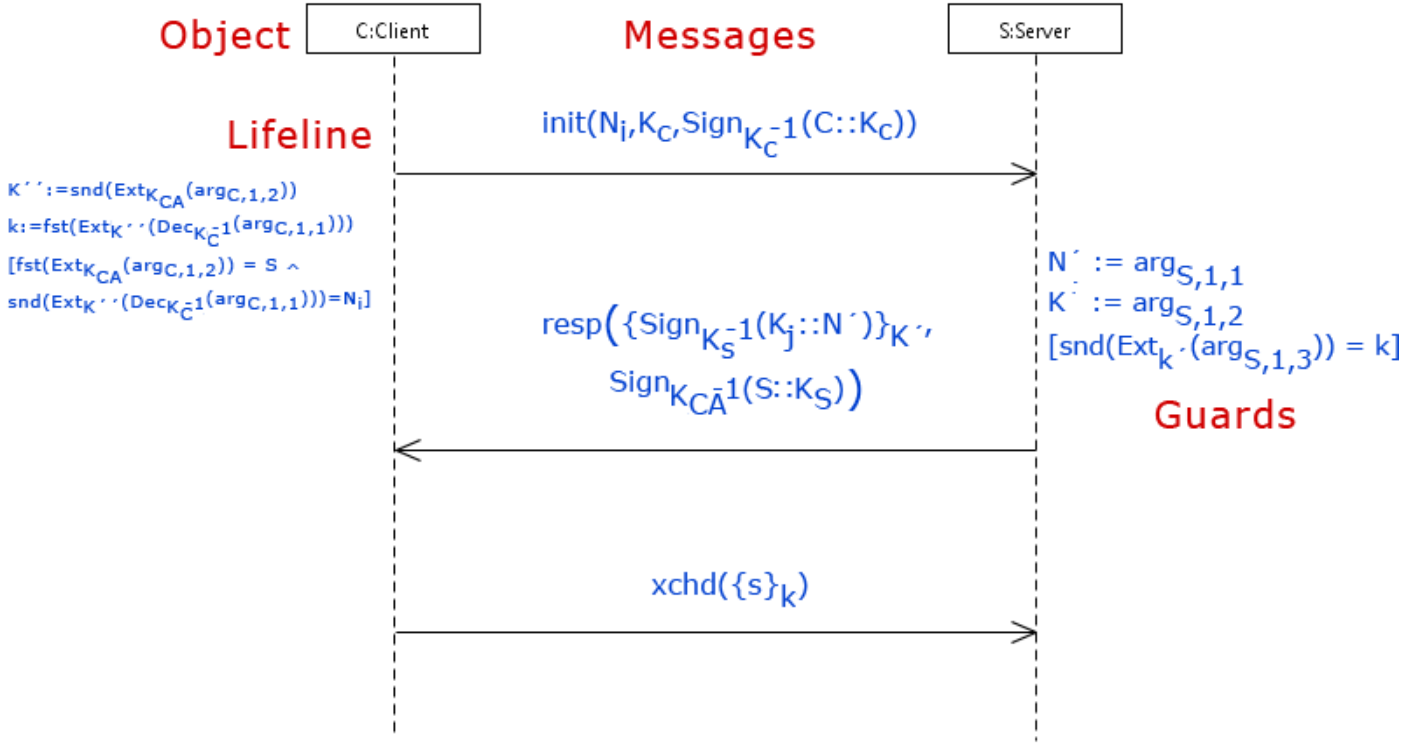
UML im Schnelldurchlauf
Sequenzdiagramm

- Diese zentrale Frage beantwortet das Diagramm:
  - Wer tauscht mit wem welche Informationen in welcher Reihenfolge aus?
- Diese Stärken hat das Diagramm:
  - stellt detailliert den Informationsaustausch zwischen Kommunikationspartnern dar
  - sehr präzise Darstellung der zeitlichen Abfolge auch mit Nebenläufigkeiten
  - Schachtelung und Flusssteuerung (Bedingungen, Schleifen, Verzweigungen) möglich

UML im Schnelldurchlauf
Sequenzdiagramm

- Häufige Anwendungsfälle
  - Die Abfolge der Nachrichten ist wichtig.
  - Die durch Nachrichten verursachten Zustandsübergänge sind kaum relevant.
  - Die Interaktionen sind kompliziert.
  - Die strukturelle Verbindung zwischen den Kommunikationspartner ist nicht relevant.
  - Es stehen Ablaufdetails im Vordergrund.

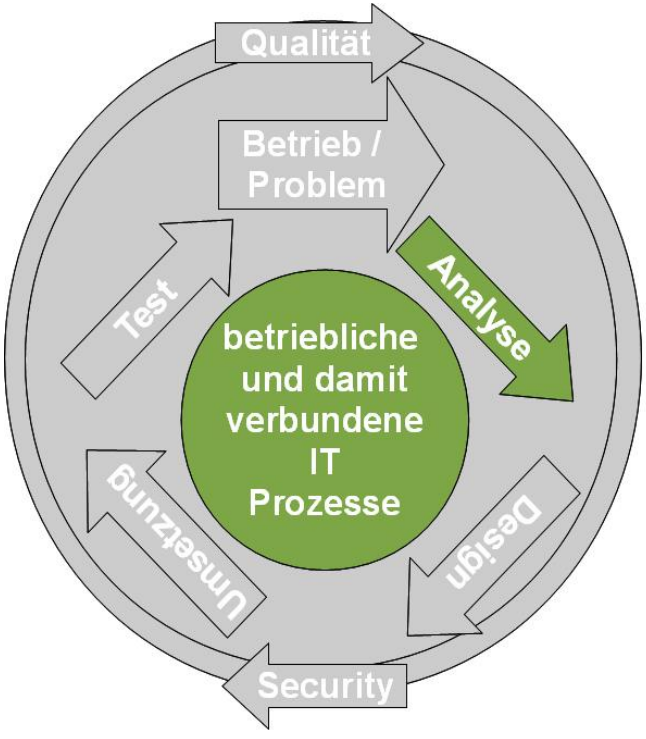
UMLsec Beispiele
Sequenzdiagramm



Verdeutlicht die Interaktion zwischen Objekten und Komponenten per Nachrichtenaustausch.

Zusammenfassung

- Diese Folien beinhalteten:
  - UML & Metaebenen
  - UML Profil
  - UML Diagramme
    - Klassendiagramm
    - Verteilungsdiagramm
    - Paketdiagramm
    - Aktivitätsdiagramm
    - Anwendungsfalldiagramm
    - Zustandsdiagramm
    - Sequenzdiagramm

Einordnung
3.0 MDA

- Geschäftsprozessmodellierung
- Process-Mining
- Modellbasierte Entwicklung sicherer Software
  - Model-Driven Architecture