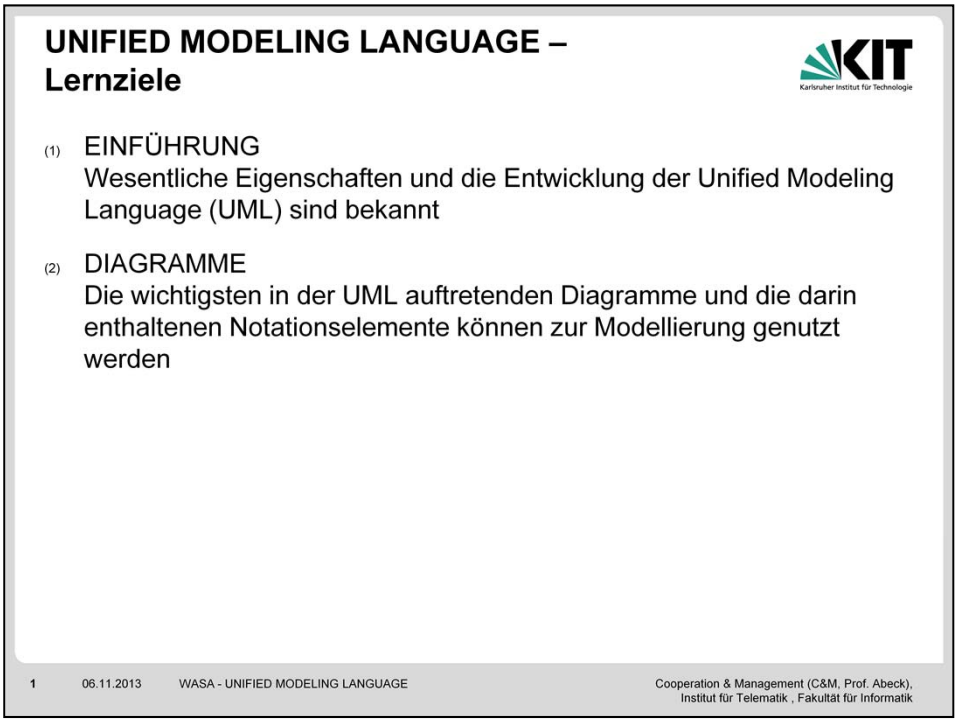


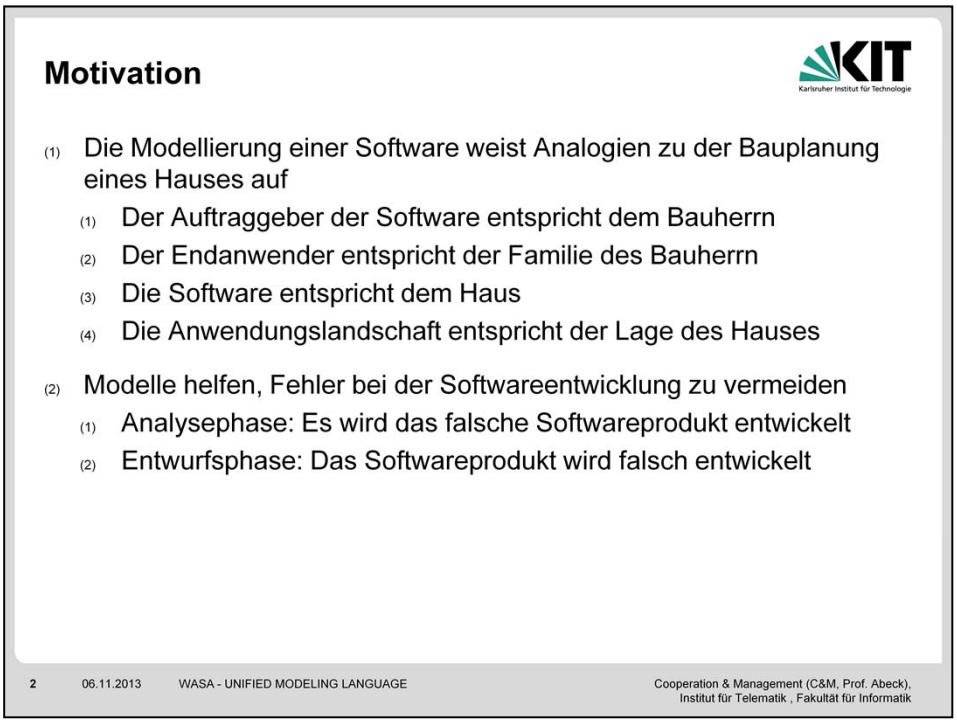
(1) Mit diesem Lernziel werden die Hintergründe zu der Modellierungssprache (Motivation, Geschichte, Entwicklung) abgehandelt.

(2) Ein Diagramm stellt eine bestimmte Sicht auf das Modell dar. Unter einem Notationselement wird ein einzelner Bestandteil eines Diagramms verstanden.

UML Unified Modeling Language

Hauptqellen:

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.
[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[Ke06:13]

(1) Der Architekt klärt mit dem Bauherrn die Größe und sonstigen Details des Hauses, bevor dieses gebaut wird. Hierzu gehören auch Fragen, wie das Haus gebaut werden soll (Fundament, Dachziegel, Anzahl Bauarbeiter und Arten von einzusetzenden Maschinen).

(1.2) Die Nutzer der Software entsprechen den Bewohnern des Hauses.

(1.3) Unter dem Begriff der Software wird hier das lauffähige Softwaresystem verstanden, zu dem auch eine geeignet Infrastruktur (Hardware) und Systemsoftware gehört, die zur Ausführung der Software benötigt wird.

(1.4) Die Anwendungslandschaft sind die angrenzenden Softwaresysteme, in die sich die zu entwickelnde Software integrieren muss.

Die Planung einer Anwendungslandschaft eines Unternehmens (engl. enterprise architecture) wird auch mit der Planung einer Stadt verglichen.

(2) Der Einsatz von Modellen und einer Modellierungssprache bietet umso größere Vorteile je größer das Projekt ist.

(2.1) In der Analyse wird ermittelt, WAS entwickelt werden soll. Hierzu gehören:

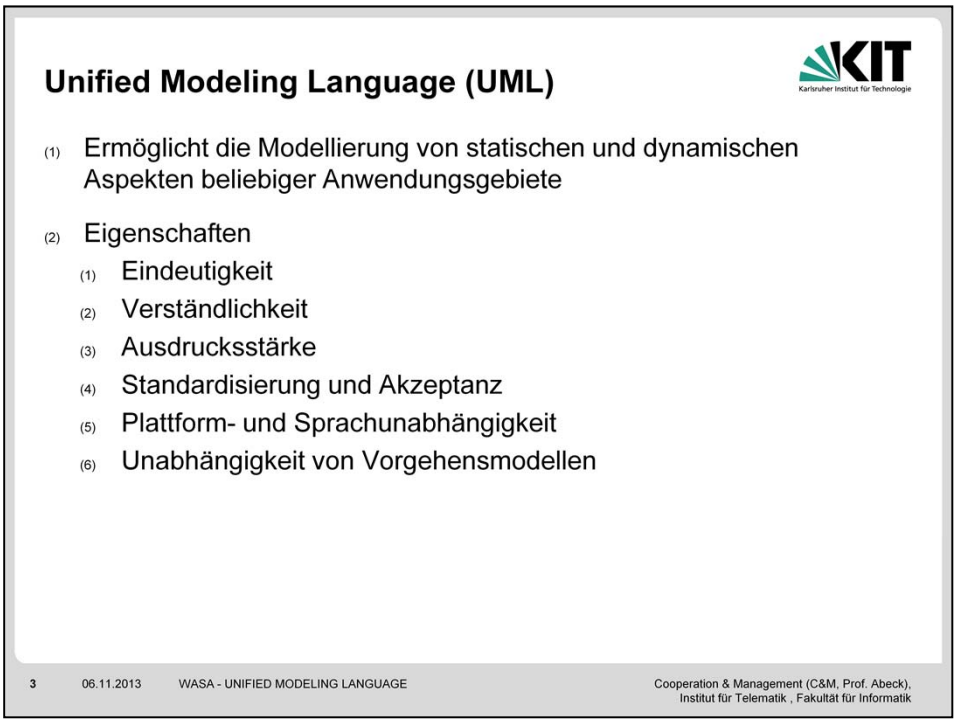
- Die von der Software zu unterstützenden Aspekte (Anwendungsfälle)
- Die zu leistenden Funktionen
- Nicht-funktionale Anforderungen, wie z.B. Antwortzeiten, Sicherheit

(2.2) Im Mittelpunkt des Entwurfs steht die Frage nach dem WIE:

- Welche Softwarearchitektur ist zur Strukturierung des Systems geeignet?
- Wie ist die Benutzeroberfläche zu gestalten?
- Welche Programmiersprachen sind am geeignetsten?
- Welche Software-Qualitätssicherungsmaßnahmen sind zu ergreifen?

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.

.

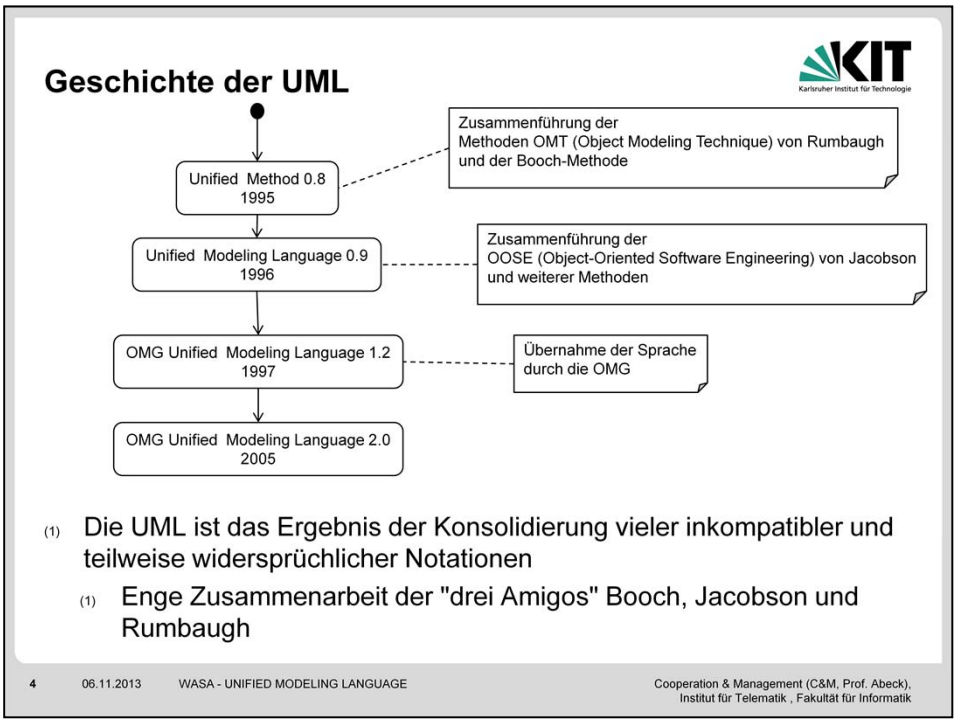


[Ke05:15]
(1) UML ist insbesondere nicht auf die Softwareentwicklung beschränkt.
Die Modellierung erfolgt mittels Diagrammen und Notationselementen.

(2) Die Eigenschaften können gleichzeitig als die Vorteile der UML angesehen werden.
(2.1) Die Eindeutigkeit wird durch eine präzise Semantik der Notationselemente in Form eines Metamodells erreicht (gilt ab der Version 2 von UML).
(2.2) Die Verständlichkeit wird erreicht durch eine grafische Visualisierung und den Einsatz von Diagrammen, die differenzierte Sichtweise auf das zu modellierende System ermöglichen.
(2.3) Eine große Breite an Notationselementen deckt nahezu alle Details ab.
(2.4) Der Object Management Group (OMG), die die UML standardisiert, gehören fast 1000 Unternehmen an, darunter alle führende Softwareunternehmen.
(2.5) Die Arbeiten sind zwar ausgerichtet auf objektorientierte Sprachen, sind aber auch für prozedurale Sprachen einsetzbar.
(2.6) Liefert nur "Werkzeuge" zur Spezifikation, überlässt dem Softwareentwickler aber, wie er diese am effizientesten einsetzen kann.

OMG	Object Management Group
UML	Unified Modeling Language

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



[Ke06:17]
Die Abbildung zeigt die Geschichte der UML als ein (vereinfachtes) Zustandsdiagramm.

(Unified Modeling Language 0.9) Zwischen 0.9 und 1.2 wurden die Versionen 1.0 und 1.1 im Jahr 1997 veröffentlicht.

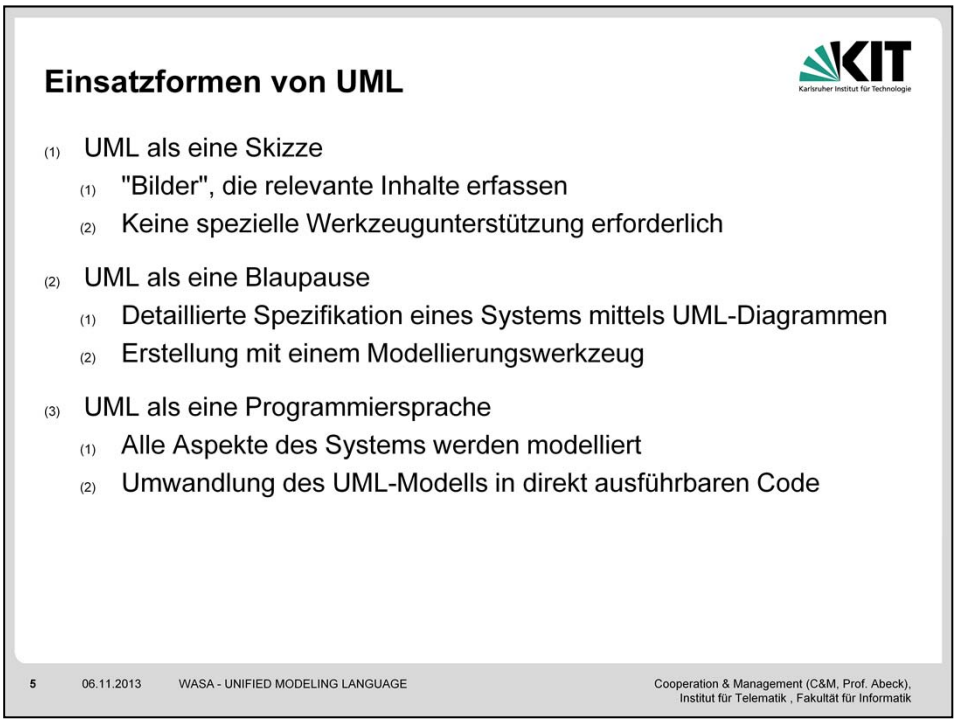
(OMG Unified Modeling Language 1.2) Diese Version wurde wegen Rechtsstreitigkeiten bei der Übergabe der Copyrights an die OMG nie veröffentlicht. Danach folgten die Versionen 1.3 in 1999, 1.4 in 2001 und 1.5 in 2003.

(OMG Unified Modeling Language 2.0) Der Sprung von der Version UML 1.X (genauer: 1.5) zur UML 2.0 war umfassend [:19]:
- Vollständige Überarbeitung verschiedener Diagramme, wie z.B. dem Aktivitätsdiagramm
- Stärkere Berücksichtigung des Zeitaspekts, z.B. durch ein ganz neues Timing-Diagramm
- Bereitstellung eines präzisen Metamodells als Grundlage für die modellgetriebene Entwicklung

(1) Die wichtigsten in der UML zusammengeführten Methoden sind:
- Object Modeling Technique (OMT) von James Rumbaugh
- Booch-Methode von Grady Booch (liefert u.a. Beiträge zu den Klassendiagrammen)
- Object-Oriented Software Engineering (OOSE) von Ivar Jacobson (beinhaltet u.a. die Anwendungsfälle)
- Object-Oriented Analysis (OOA) von Peter Coad und Edward Yourdon

OMT	Object Modeling Technique
OOA	Object-Oriented Analysis
OOSE	Object-Oriented Software Engineering

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



[MH06:13]
Das Einsatzspektrum von UML ist sehr breit – von der einfachen Erstellung von Skizzen bis hin zum vollständigen Ersatz von höheren Programmiersprachen.

(1) Der sehr beschränkte Einsatz zur graphischen Skizzierung von Teilaspekten des zu entwickelnden Softwaresystems wird in den meisten Industriebereichen eingesetzt, in denen sich die Qualitätsanforderungen auf einem durchschnittlichen Niveau befinden und ein stärkerer Einsatz von Modellierungstechniken zu aufwändig und ineffizient erscheint.
(1.1) Diese Bilder dienen zur Diskussion und sind sog. "Wegwerf-Artefakte".
(1.2) Erstellung auf Papier, z.B. auf einem Whiteboard.

(2) Diese intensivere Nutzung der UML erfolgt in Bereichen wie der Medizin oder Verteidigung, in denen ein hohes Maß an Qualität erforderlich ist.
(2.1) Die Modelle sind durch Vorwärts- und Rückwärts-Entwicklung (engl. forward and reverse engineering) mit dem Code synchron zu halten.
(2.2) Es handelt sich um wesentliche Artefakte der Softwareentwicklung, also nicht nur Bilder bzw. Wegwerf-Artefakte.

(3) Diese Einsatzform der UML ist heute noch weitestgehend eine Vision.
(3.1) Bei der Vorwärts-Entwicklung (engl. forward engineering) werden nur Teile des Codes erzeugt.
(3.2) Dies erfolgt durch Transformation oder Code-Erzeugung. Es kann eine Auslieferung in verschiedene Umgebungen erfolgen.

[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.

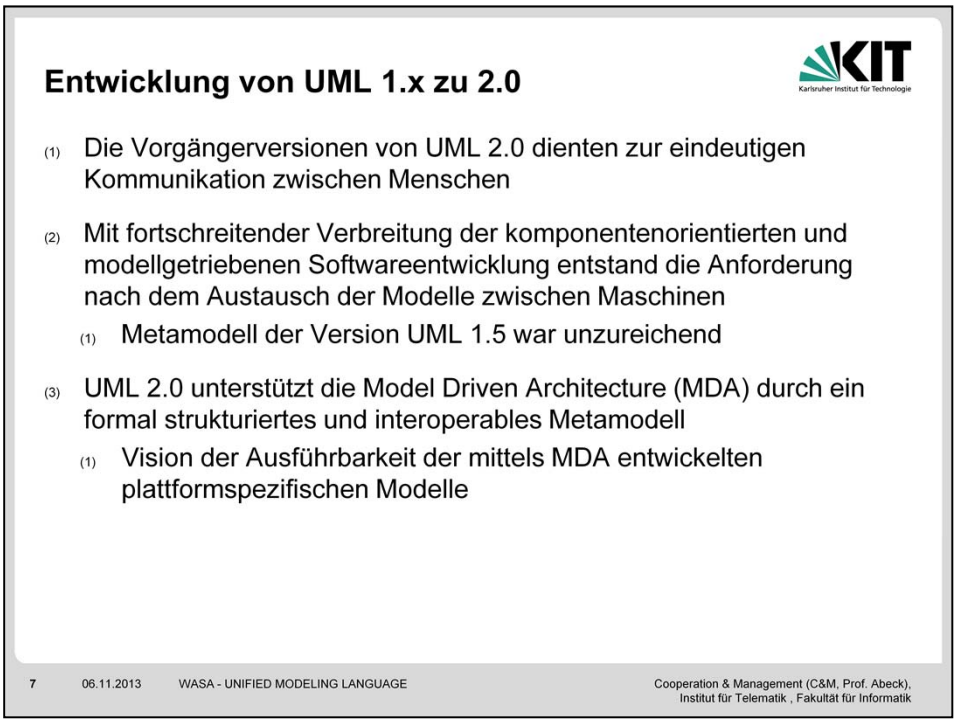
LZ EINFÜHRUNG – ÜA ÜBERBLICK

(1) Welche Fehler sollen durch den Einsatz von Modellen verhindert werden
   (1) in der Analysephase?
   (2) in der Entwurfsphase?

(2) Wer sind die "drei Amigos"?

(3) Welche Ansätze wurden in der UML vereinheitlicht (engl. unified) und auf welche Eigenschaften hat das einen unmittelbaren Einfluss genommen?

(4) Wie wird UML heute meist eingesetzt und was ist das visionäre Ziel?



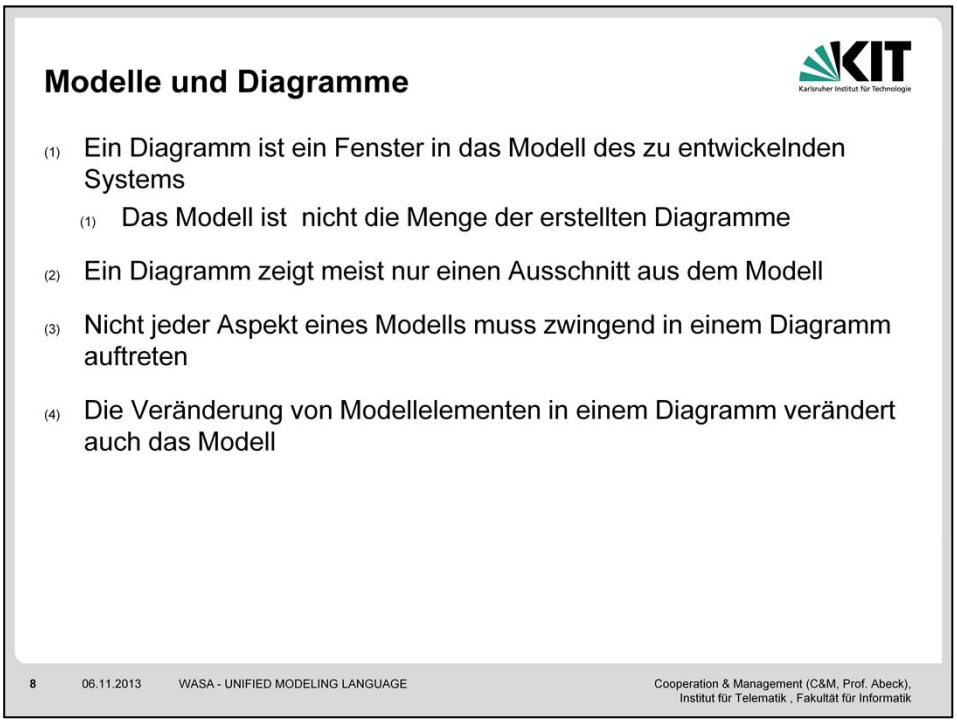
[MH06:9]
(1) Die Vorgängerversionen werden als "UML 1.x" bezeichnet.
Die in UML 1.x beschriebenen Modelle sollten den Kern eines Designs erfassen und die funktionalen Anforderungen an die Software abbilden [:9].

(2) Ironischerweise waren die den Sprachen UML 1.x zugrunde gelegten Notationsregeln und das Metamodell nicht formal genug, um einen Maschine-zu-Maschine-Austausch zu ermöglichen.
(2.1) Das Metamodell war historisch gewachsen, wirkte daher zerstückelt und zu komplex.

(3) Die Model Driven Architecture (MDA) stellt ein Rahmenwerk zur Entwicklung von plattformunabhängigen Modellen (engl. Platform-Independent Models PIM) zur Verfügung, die mittels Transformationen (die auf Metamodell-Ebene spezifiziert werden) in plattformspezifische Modelle (engl. Platform-Specific Models PSM) übertragen werden.
(3.1) Diese Vision wird durch das sog. Executable UML verfolgt, dem eine UML-Maschine zugrunde liegt, die hinreichend genau spezifizierte UML-Modelle auszuführen gestattet.

MDA	Modsel Driven Architecture
PIM	Platform-Independant Model
PSM	Platform-Specific Model

[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



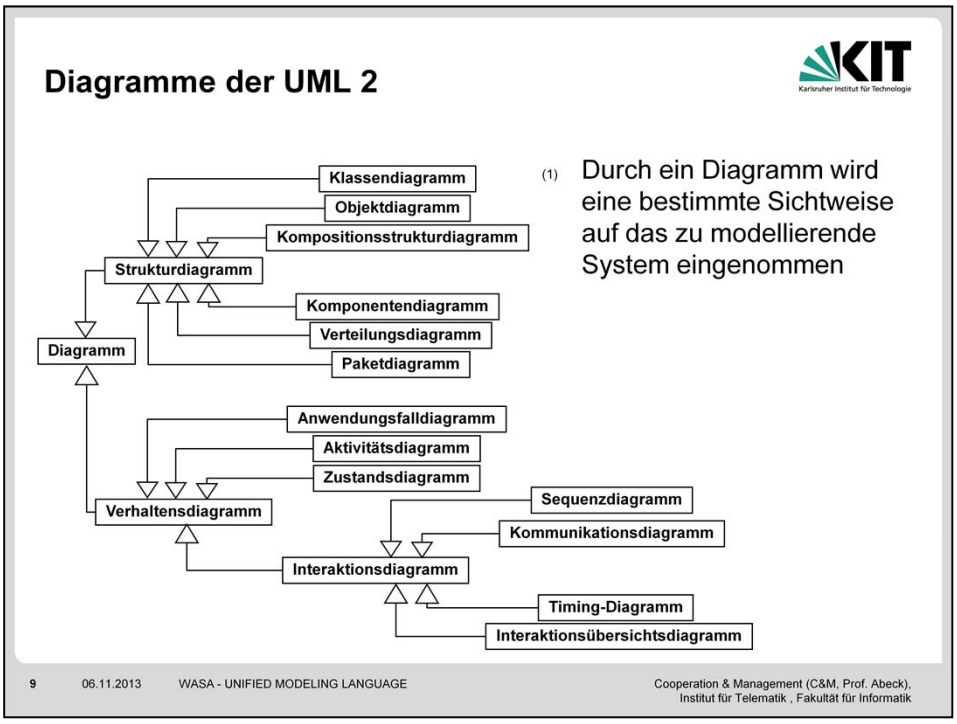
[MH06:12]
(1) Ein UML-Anfänger sieht zunächst nur die Diagramme.
(1.1) Das Modell befindet sich "hinter" den Diagrammen.
Die Diagramme entsprechen einer Art Präsentationsschicht, während das eigentliche Modell in einer darunterliegenden Datenschicht liegt.

(2) Hierdurch kann die Komplexität des Modells beherrschbar gemacht werden, indem verschiedene Modellaspekte auf mehrere Diagramme aufgeteilt werden.

(3) Das Modell ist die Sammlung aller das System beschreibenden Elemente und deren Beziehungen. Solche Elemente und Beziehungen können auch ohne Verwendung eines speziellen Diagramms spezifiziert werden.

(4) Eine Änderung im Diagramm ist zunächst nur eine Änderung der Sicht (auf Präsentationsebene), die aber dann auf das Modell (auf Datenebene) propagiert wird.

[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[Ke06:20]
Von der Version UML 1.5 auf UML 2 hat die OMG die Diagrammarten neu angeordnet und teilweise vollständig überarbeitet, wodurch die Sprache konsistenter und (trotz höherer Mächtigkeit) einfacher geworden ist [:19].

(1) Durch Diagramme lässt sich die Komplexität eines Modells geeignet zerlegen, was zur Verständlichkeit von UML beiträgt.

(Strukturdiagramm) engl. structure diagrams: Modellieren statische, zeitunabhängige Elemente des Systems.
(Verhaltensdiagramm) engl. behaviour diagrams: Modellieren die dynamischen Aspekte, das Verhalten des Systems und seiner Komponenten.

Nachfolgend erfolgt exemplarisch die Beschreibung von vier konkreten in UML bereitgestellten Diagrammtypen (jeweils ein statisches und ein dynamisches Diagramm im Wechsel):

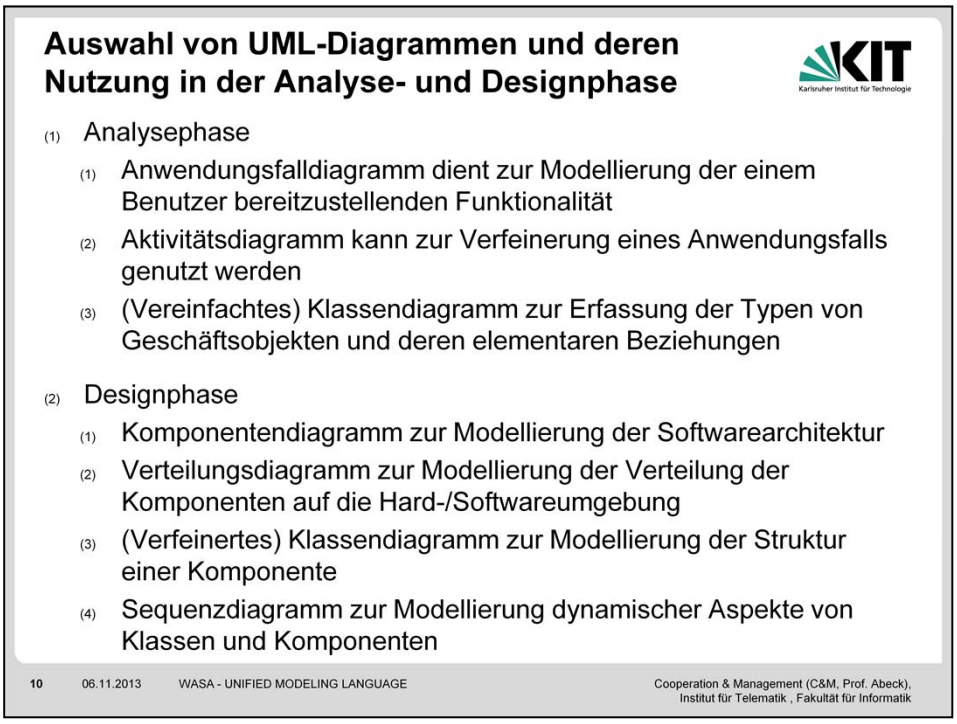
(Klassendiagramm) Beinhaltet die statischen Strukturbestandteile eines Systems, deren Eigenschaften und Beziehungen [:21].

(Anwendungsfalldiagramm) Anwendungsfälle stellen eine Menge von Aktionen dar, die ein Akteur dynamisch (im zeitlichen Verlauf) während der Interaktion mit dem System durchläuft.

(Komponentendiagramm) Beschreibt die Organisation von und die Abhängigkeiten zwischen Komponenten, die ersetzbare modulare Bestandteile eines Systems darstellen [:147]. Ein Komponentendiagramm wird insbesondere zur Beschreibung einer logischen Softwarearchitektur genutzt.

(Aktivitätsdiagramm) Beschreiben das Verhalten einer Klasse oder einer Komponente und bedienen sich dabei eines Kontroll- und Datenflussmodells.

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



Die UML und die darin bereitgestellten Diagramme sind unabhängig von einem Softwareentwicklungsprozess und machen keine Aussage darüber, in welcher Reihenfolge und unter welchen Randbedingungen ein Diagramm-Typ eingesetzt werden soll.

Unabhängig davon hat sich in der Praxis ein Vorgehen etabliert, welche Diagramm-Typen üblicherweise innerhalb der ersten zwei Phasen des Softwareentwicklung genutzt werden können bzw. sollten.

(1.1) Ein Anwendungsfalldiagramm ist üblicherweise der erste UML-Diagrammtyp, der im Softwareentwicklungsprozess verwendet wird.
(1.2) Hinter einem Anwendungsfall steckt ein Stück Geschäftsprozess, der die Interaktion zwischen den beteiligten Rollen und dem System fokussiert.
(1.3) Das Klassendiagramm ist (im Gegensatz zu dem in der Designphase erstellten Klassendiagrammen) sehr einfach gehalten, damit der Kunde bzw. zukünftige Anwender des Systems den Inhalt dieses Geschäftsklassendiagramms verstehen kann.

(2.1) Eine Komponente stellt den zentralen Baustein einer Softwarearchitektur dar.
(2.2) Ein Verteilungsdiagramm (engl. deployment diagram) behandelt den physischen Aspekt einer Softwarearchitektur, indem die Laufzeitumgebung des Softwaresystems spezifiziert wird.
(2.3) Das Klassendiagramm kann so fein und umfassend spezifiziert sein, dass eine (halb-) automatische Transformation des Codes einer objektorientierten Sprache erfolgen könnte.
(2.4) Die dynamischen Diagramme können immer nur gewisse (besonders relevante oder kritische) Aspekte der Software erfassen.

Die hier beschriebene "Best Practice" bestimmt die Auswahl und die Reihenfolge der nachfolgend detaillierter eingeführten UML-Diagrammtypen:
(i) Anwendungsfalldiagramm
(ii) Aktivitätsdiagramm
(iii) Klassendiagramm
(iv) Komponenten-/Verteilungsdiagramm
(v) Sequenzdiagramm

LZ EINFÜHRUNG – ÜA UML 2

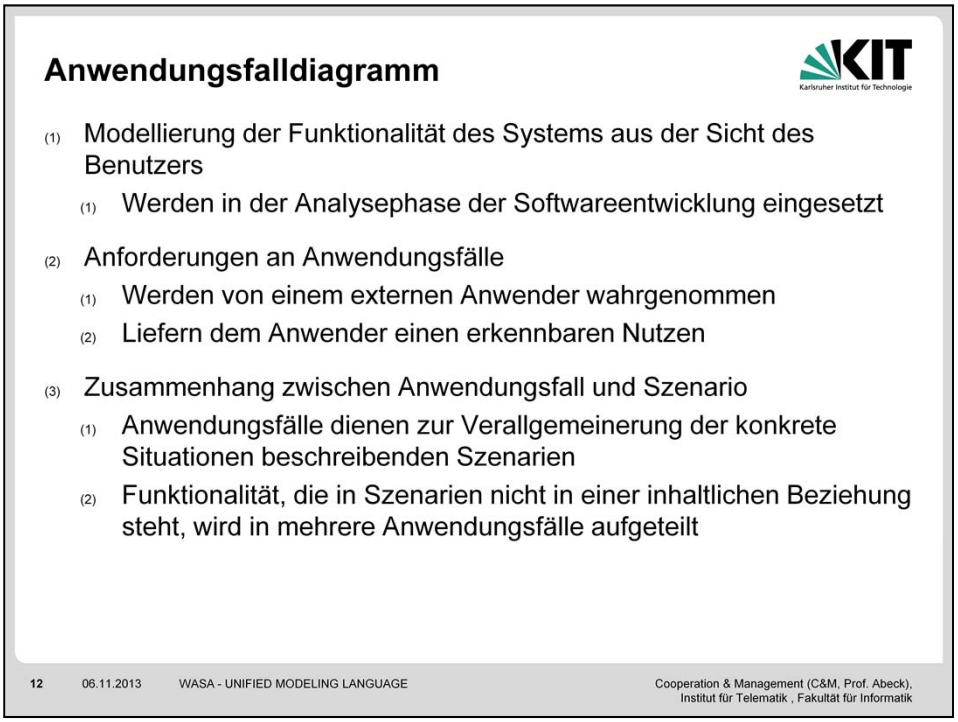
(1) Welches elementare Defizit hatten die Vorgängerversionen von UML2?

(2) Welchen Beitrag leistet die UML2 zur Model Driven Architecture (MDA)?

(3) Wie ist der Zusammenhang zwischen Modell und Diagramm?

(4) Strukturdiagramme sind unabhängig von der _____

(5) Die Charakteristik von Strukturdiagrammen und Verhaltensdiagrammen ist an jeweils einem Beispiel eines konkreten Diagrammtyps zu erklären

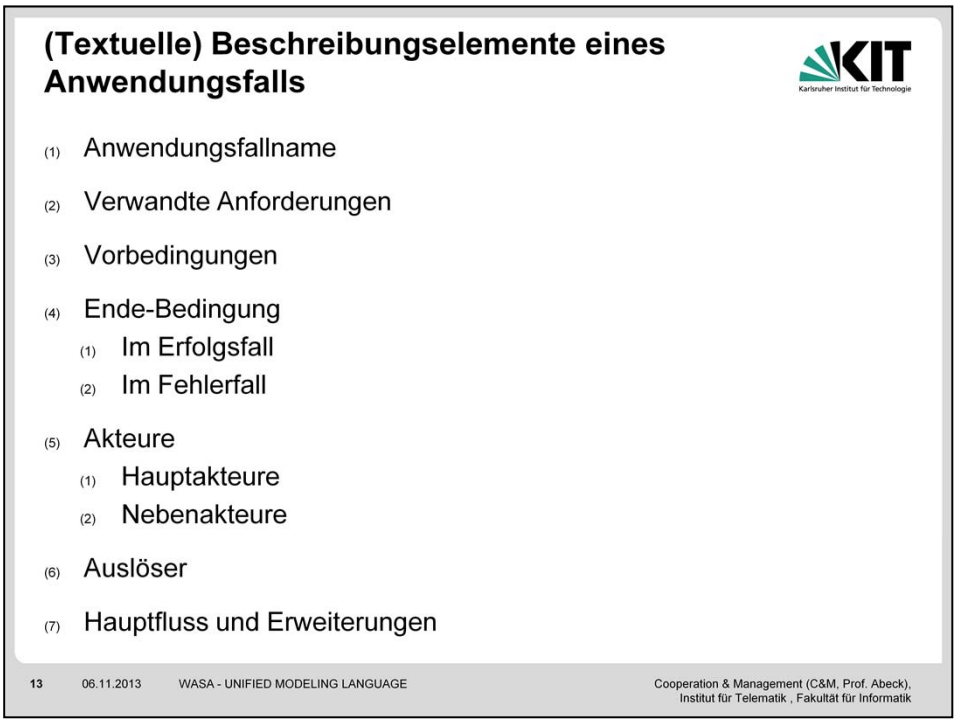


[Ke06:197]
(1) Es wird eine Black-Box-Sicht eingenommen und die Funktionalitätsbeschreibung erfolgt auf einem hohen Abstraktionsniveau.
(1.1) Anwendungsfalldiagramme sind üblicherweise die ersten im Entwicklungsprozess eingesetzten UML-Diagramme.

(2.1) Wird auch als externes Verhalten bezeichnet [BD04:44]. Ein Anwendungsfall verdeutlicht, dass eine Funktionalität bereitgestellt wird, aber nicht, wie das System diese erbringt [Ke06:201].
(2.1) Ein Anwendungsfall wird von genau einem Akteur initiiert. Durch die Initiierung nehmen die beteiligten Akteure gewisse Funktionalität des Systems in Anspruch.
(2.2) Zudem sollte ein Anwendungsfall von genau einem Akteur initiiert werden.

(3) In der Kurseinheit BEISPIEL KIT-SMART-CAMPUS werden sog. Szenarien benutzt, die mit Anwendungsfällen in Zusammenhang gebracht werden können.
(3.1) Die Verallgemeinerung wird auch dadurch deutlich, dass anstelle konkreter Personen (Akteur-Instanzen) in Anwendungsfällen die Akteure auftreten.
(3.2) Anwendungsfälle sollten eine kohärente, weitestgehend abgeschlossene und unabhängige Funktionalität beschreiben [BD04:159].
Hinsichtlich dieser Eigenschaften bestehen Ähnlichkeiten zu den später behandelten (Software-) Services.

[BD04] Bernd Bruegge, Allen H. Dutoit: Object-Oriented Software Engineering Using UML, Patterns and Java, Pearson Prentice Hall, 2004.
[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



[MH06:28]
Zu jedem Anwendungsfall sollte eine strukturierte textuelle Beschreibung vorliegen, durch die ein Systemdesigner die Systembelange in angemessener Form verstehen kann. Ohne diese zusätzliche Information ist ein Anwendungsfall wenig hilfreich.

Die folgenden Beschreibungselemente geben einen Hinweis darauf, welche Informationen benötigt werden.

(1) Ein Anwendungsfallname sollte syntaktisch einheitlich aufgebaut sein.
Empfehlung: <Verb> <Subjekt>
Anmerkung: Alle Artefakte werden in dieser Vorlesung (und auch durchgängig in allen bei C&M durchgeführten Softwareentwicklungsarbeiten) in englischer Sprache benannt.

(2) Hier sollten Hinweise darauf erfolgen, welche (ggf. in einem Lastenheft aufgeführte) Anforderungen der Anwendungsfall vollständig oder teilweise erfüllt.

(3) Was muss geschehen, damit der Anwendungsfall angestoßen werden kann.

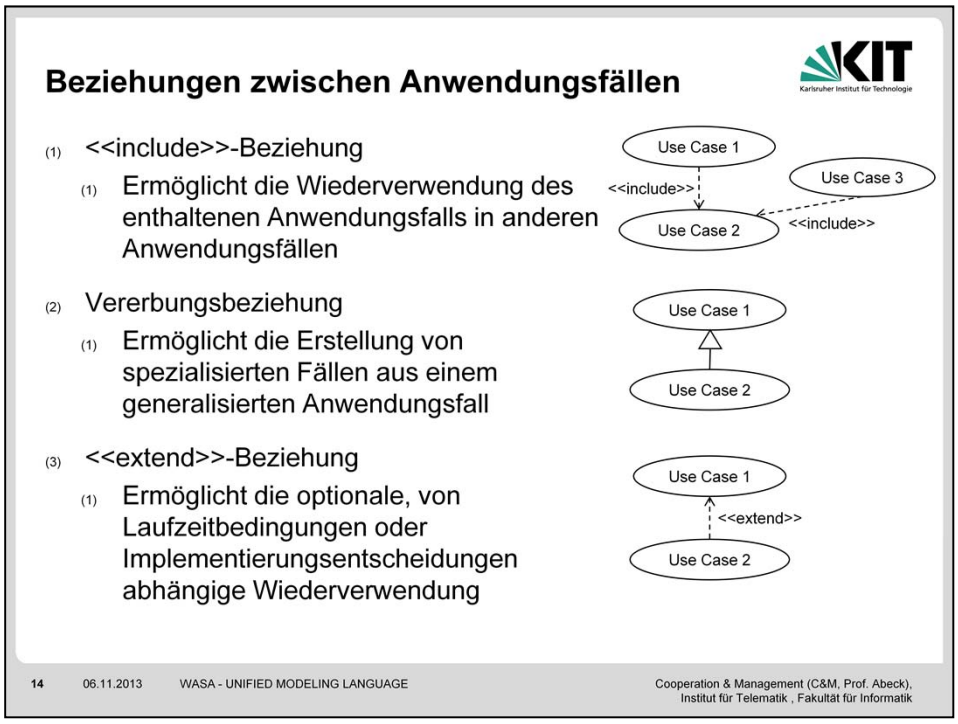
(4) Der nach erfolgreicher bzw. fehlerhafter Ausführung des Anwendungsfalls erreichte Systemzustand.

(5) Hauptakteure (engl. primary actor) lassen sich von Nebenakteuren (engl. secondary actor) dadurch unterscheiden, dass diese den Anwendungsfall auslösen oder direkt Informationen vom Anwendungsfall erhalten.

(6) Das von einem (Haupt-) Akteur ausgelöste Ereignis, durch das der Anwendungsfall ausgeführt wird.

(7) Hauptschritte und alternative bzw. ergänzende Schritte, die bei der Ausführung eines Anwendungsfalls durchgeführt werden.

[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



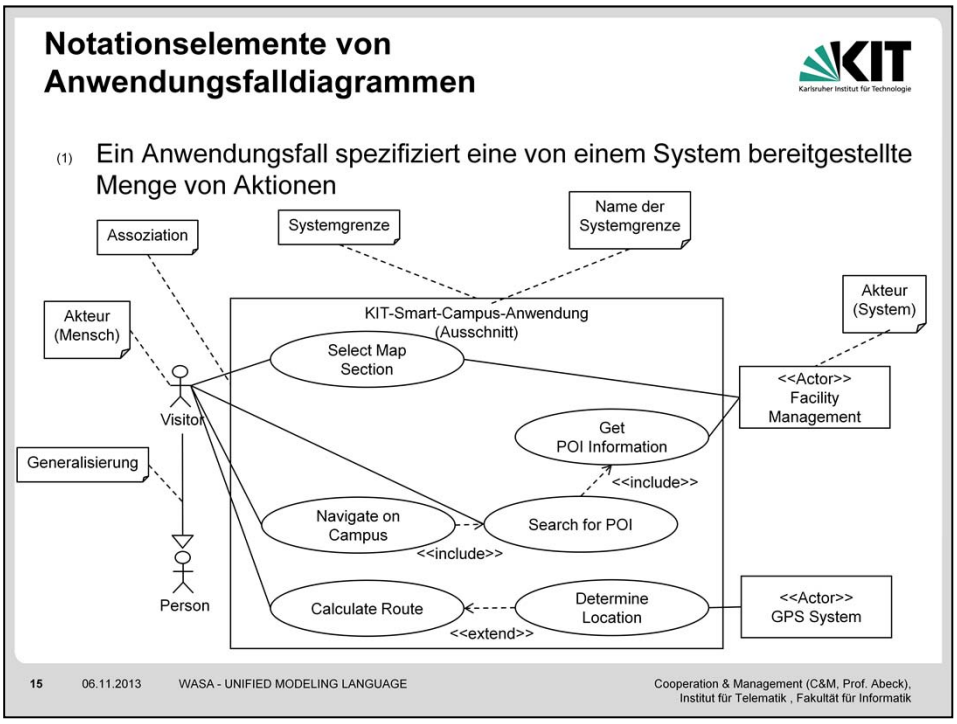
[MH06:30]
Bei der Erfassung von Anwendungsfällen kann der Fall auftreten, dass gewisse Ähnlichkeiten in verschiedenen Anwendungsfällen auftreten oder dass Anwendungsfälle in verschiedenen Situationen unterschiedlich ablaufen. Diese Aspekte lassen sich durch die Modellierung von Beziehungen zwischen Anwendungsfällen beschreiben.

(1) Die <<include>>-Beziehung drückt aus, dass der enthaltene Anwendungsfall (im Beispiel "Use Case 2") vollständig in dem diesen enthaltenden Anwendungsfall ("Use Case 1") auftritt.
(1.1) Der "Use Case 2" wurde ggf. aus dem "Use Case 1" genau deshalb herausgezogen, weil festgestellt wurde, dass der gleiche Ablauf auch in "Use Case 3" auftritt und daher darin wiederverwendet werden kann.

(2) Auch hier geht es wie bei der <<include>>-Beziehung um Wiederverwendung, jetzt aber wird nicht ein kleiner enthaltener Teil, sondern ein verallgemeinerbarer Teil (im Beispiel "Use Case 1") wiederverwendet und durch den verfeinerten Anwendungsfall ("Use Case 2") im Hinblick auf spezielle Situationen erweitert.
(2.1) Man spricht hier von Anwendungsfall-Verallgemeinerung (engl. generalization) oder –Vererbung (engl. inheritance).

(3) Die <<extend>>-Anwendungsfallbeziehung gehört zu den am meisten diskutierten UML-Modellierungskonzepten. Die hiermit assoziierte Form der Erweiterung unterscheidet sich deutlich von der aus einer Vererbungsbeziehung resultierenden Erweiterung, da es sich nicht um eine "Verfeinerung", sondern um eine "bedingte Ergänzung" eines völlig neuen Anwendungsfalls zu dem bestehenden Anwendungsfall handelt.
Wie bei der <<include>>-Beziehung wird der erweiternde Anwendungsfall (im Beispiel: "Use Case 2") vollständig in den erweiterten Anwendungsfall ("Use Case 1") einbezogen.
(3.1) Solche Erweiterungen betreffen insbesondere Abläufe, die in Ausnahme- oder Fehlerfällen auftreten.

[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[Ke06:197]
(1) Diese abgeschlossene Aktionenmenge muss für einen oder mehrere Benutzer einen erkennbaren Nutzen erbringen und kann durch weitere Verhaltensdiagramme verfeinert und präzisiert werden [:197].

(Systemgrenze) Umfasst ein System, das die benötigten Anwendungsfälle bereitstellt und mit dem die Anwender interagieren [:198].

(Assoziation) Zu der Assoziation zwischen Rolle und Anwendungsfall können Kardinalitäten angegeben werden. Eine Assoziation ohne Kardinalitätsangabe entspricht einer 1-zu-1-Assoziation [Kecher06:208].

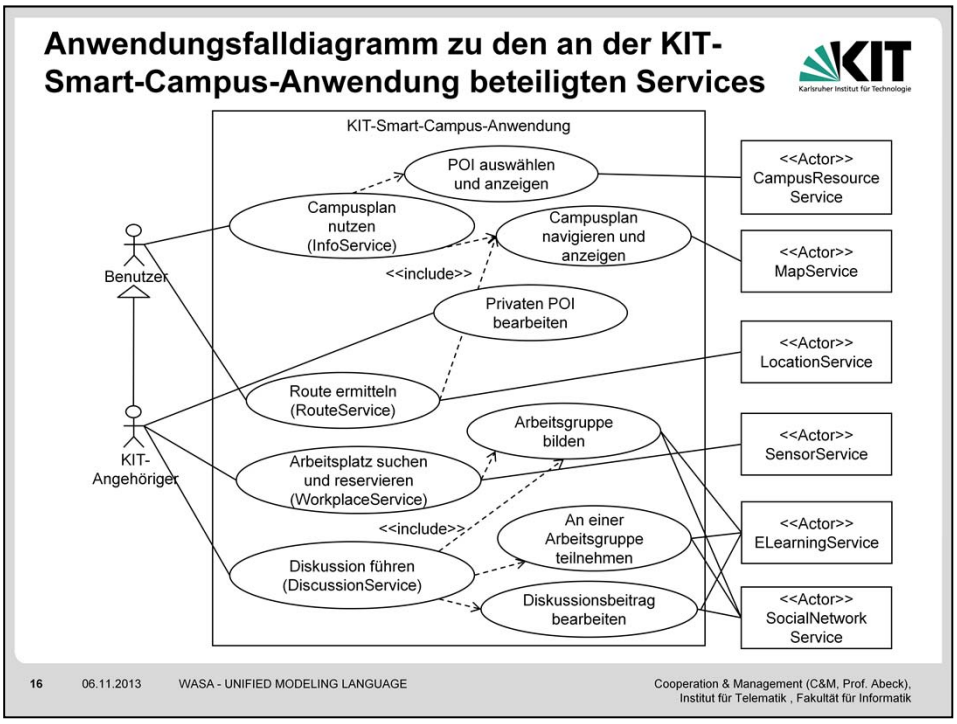
(Akteur) Modelliert einen Typ oder eine Rolle, die ein externer Benutzer oder ein externes System während der Interaktion mit einem System einnimmt [:199].
Der Stickman ist wie <<Actor>> ein Stereotyp (spezifizieren die Art eines Notationselements und geben eine Auskunft über den Zweck des Elements im Modell [:85]).

(<<include>>) einbindend ------> eingebunden: Unbedingte Einbindung der Funktionalität eines eingebundenen Anwendungsfalls in den einbindenden Anwendungsfall
Im Beispiel: Für die Navigation auf dem Campus ist die Suche nach POIs zwingend erforderlich. Für die Suche eines POI ist zwingend erforderlich, dass die Information zu POIs gelesen werden muss. Die Suche nach POIs kann auch direkt vom Benutzer angestoßen werden.

(<<extend>>) erweiternd ------> erweitert: Bedingte Einbindung der Funktionalität des erweiternden Anwendungsfall in den erweiterten Anwendungsfall
Im Beispiel: Für gewisse Start- bzw. Endpunkte der zu berechnenden Route ist zuvor deren Lokation (z.B. mittels GPS) zu ermitteln, z.B. wenn die Route vom aktuellen Standort zu einem vom Benutzer einzugebenden Zielort zu bestimmen ist.

GPS	Global Positioning System
KIT-SC	KIT-Smart-Campus
POI	PointÓfInterest

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



Die KIT-Smart-Campus-Anwendung wird als ein serviceorientiertes Softwaresystem entwickelt. Ein Anwendungsfalldiagramm kann dazu verwendet werden, die dem Benutzer zur Verfügung zu stellenden Services und die benötigten Services auf einer hohen Abstraktionsebene in übersichtlicher Form darzustellen [AH+13].

(InfoService) Der Service ermöglicht das Anlegen von privaten Points-of-Interest, wodurch eine auf die Informationsbedürfnisse des Benutzers angepasste, individualisierte Campus-Karte entsteht.
(RouteService) Ergänzt wird der InfoService um einen RouteService, durch den die Bestimmung des eigenen Standorts und die Route zwischen zwei beliebigen Punkten auf dem Campus ermöglicht werden.
(WorkplaceService) Ein weiterer KIT-Smart-Campus-Service, der WorkplaceService, bietet eine Suche und Reservierung von Arbeitsplätzen für Arbeitsgruppen auf dem Campus an. Durch den Einsatz von Sensoren in den Arbeitsräumen und an den Arbeitsplätzen wird erreicht, dass die Feststellung der Anzahl an freien Arbeitsplätze in einem Raum und die Reservierung effizient und smart durchgeführt werden können.
(DiscussionService) Ein vierter Service, der in der ersten Ausbaustufe der KIT-Smart-Campus-Service-Anwendung geplant ist, ist ein DiscussionService, über den eine auf dem Campus arbeitende Arbeitsgruppe smarte Diskussionen führen kann.

(CampusResourceService) Der engste Bezug zu den am KIT bestehenden Systemen und Anwendungen ist im Zusammenhang mit der Information zum Campus, die in den POIs gehalten werden, gegeben. Der überwiegende Anteil der in der KIT-Smart-Campus-Anwendung genutzten POI-Daten muss über einen geeigneten CampusResourceService aus den am KIT betriebenen Facility-Management-System (Morada), Lehrveranstaltungs-System (CAS-Campus-Management) und ggf. weiteren Systemen übernommen werden.
(MapService) Der Campusplan selbst ist über einen MapService bereitgesellt. Hier ist festzulegen, ob dieser vom KIT bereitgestellt werden kann oder ob (zunächst) ein externer, kostenfreier MapService genutzt werden soll.
(LocationService) Dieser von externer Seite bereitgestellte Service liefert den aktuellen Standort des Benutzers, der die KIT-Smart-Campus-Services mit einem mobilen Gerät nutzt. Eine weit verbreitete zu diesem Zweck genutzte Technologie ist das Global Positioning System (GPS).
(SensorService) Dieser Service stellt über entsprechend an Räumen und Arbeitsplätzen angebrachte Sensoren (Video-Kameras, QR-Codes, RFID-Chips) Informationen über die Verfügbarkeit von Arbeitsplätzen in einem Raum zur Verfügung und ermöglicht eine einfache Reservierung der Arbeitsplätze.
(ELearningService) Bei der Unterstützung der Zusammenarbeit von Arbeitsgruppen auf dem KIT-Campus besteht eine enge Verbindung mit den am KIT betriebenen E-Learning-Systemen (z.B. Ilias).
(SocialNetworkService) Die durch den DiscussionService bereitgestellte Kollaborationsunterstützung soll die Möglichkeiten berücksichtigen, die heute soziale Netzwerke wie Facebook oder Google bieten.

[AH+13] Sebastian Abeck, Philip Hoyer, Roland Steinegger, et al.: Projekt KIT-Smart-Campus, KIT C&M – Fraunhofer IOSB – iC Consult – Kooperationsprojekt, 2013. *C&M-Teamserver > Mitglieder > 3-3.KIT-Smart-Campus > 1-2.Projektbericht*

LZ DIAGRAMME – ÜA ANWENDUNGSFALLDIAGRAMM

(1) Was muss erfüllt sein, damit eine Systemfunktionalität als ein Anwendungsfall beschrieben werden sollte?

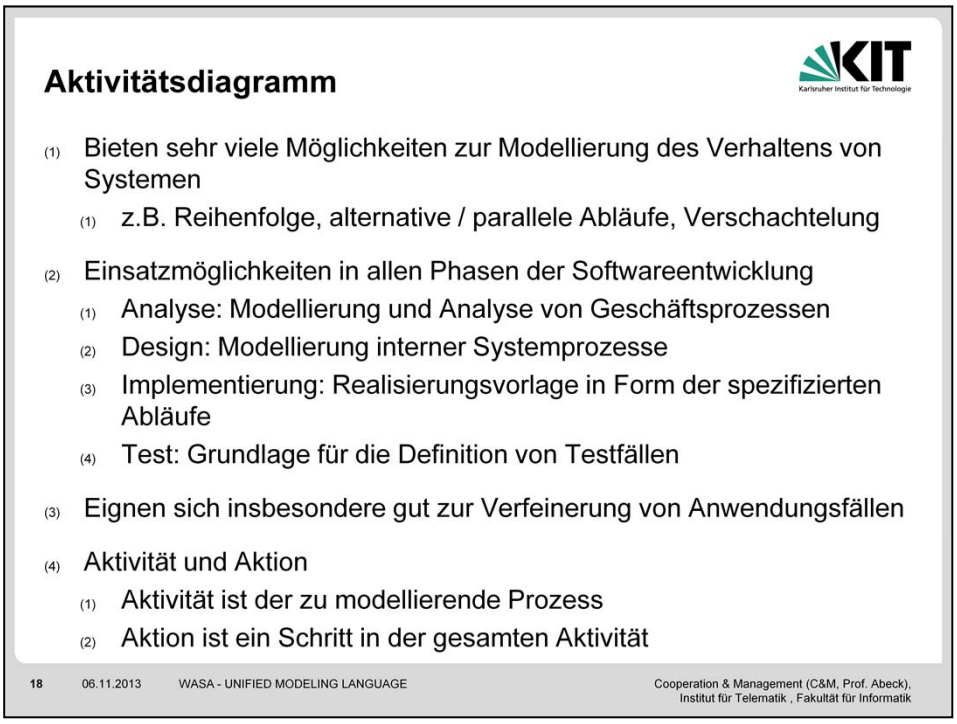
(2) Wie ist der Zusammenhang zwischen einem Szenario und einem Anwendungsfall?

(3) Welche Informationen benötigt ein Systemdesigner, um einen Anwendungsfall verstehen zu können?

(4) Welche Ähnlichkeiten und welche Unterschiede weisen die drei, zwischen Anwendungsfällen modellierbaren Beziehungstypen auf?

(5) Bezieht sich ein Anwendungsfall auf einen (Geschäfts-) Prozess oder auf ein (IT-) System? An welchem Notationselement im Diagramm ist das erkennbar?

(6) Wie können benötigte Services in einem Anwendungsfall modelliert werden?



[Ke06:213]
(1) Ein Aktivitätsdiagramm gehört zu den UML-Verhaltensdiagrammen und stellt (wie z.B. auch BPMN oder BPEL) eine Flussdiagramm-Beschreibung (engl. flow charts) dar.
(1.1) Das sind die Hauptsprachelemente von Flussdiagrammen. Weitere in Aktivitätsdiagrammen einsetzbare Sprachelemente betreffen die Festlegung von Verantwortungsbereichen oder die Ausnahmebehandlung.

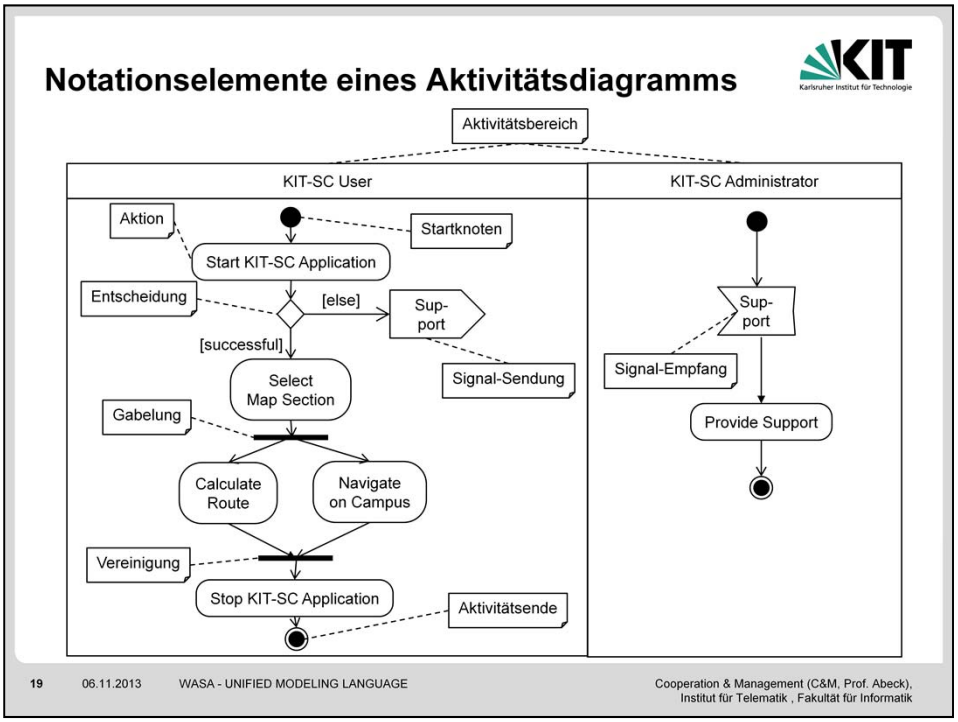
(2) In jeder Phase drücken dabei Aktivitäten ganz unterschiedliche Sachverhalte aus (z.B. in der Analyse eine Geschäftsaktivität, während im Entwurf und in der Implementierung einen Berechnungsschritt als Teil eines Algorithmus).
(2.1) Zeigen die Reihenfolge der am Geschäft beteiligten Rollen und deren Geschäftstätigkeiten sowie mögliche Alternativabläufe.
Mögliche Optimierungen werden direkt wieder in Aktivitätsdiagrammen dokumentiert.
(2.2) Das können Abläufe innerhalb eines Systems oder zwischen Systemen sein. Die Granularität der Aktivitäten kann so fein gewählt sein, dass ein Aktivitätsdiagramm Umsetzungsvorschriften komplexer Algorithmen darstellt [:214].
(2.3) An dieser Stelle könnten die Aktivitätsdiagramme im Sinne der modellgetriebenen Entwicklung automatisiert in ausführbaren Code auf der Ebene einer höheren Programmiersprache transformiert werden.
(2.4) Hierfür müssen die Aktivitätsdiagramme allerdings eine möglichst exakte Ablaufspezifikation liefern.

(3) Das entspricht der Nutzung in der Analysephase: Anwendungsfälle verbergen die internen Abläufe, die durch Aktivitätsdiagramme explizit gemacht werden.

(4) Die beiden Begriffe werden häufig fälschlicherweise synonym verwendet [MH06:46]
(4.1) Eine Aktivität besitzt daher grundsätzlich ein Anfangs- und einen Endepunkt.
(4.2) Eine Aktion hat die Eigenschaft der Atomizität, d.h. sie ist nicht weiter unterteilbar.

BPEL	Business Process Execution Language
BPMN	Business Proecess Model and Notation

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.
[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[Ke06:215]
Das Beispiel zeigt den stark vereinfachten (Geschäfts-) Prozess zur Nutzung des KIT-Smart-Campus (KIT-SC).

(Aktivitätsbereich) Gruppieren Aktivitätsknoten zu Organisationseinheiten [:219]. Der Begriff der Organisationseinheit ist weit gefasst und kann z.B. ein Unternehmen oder auch eine menschliche oder technische Rolle sein. Diese übernimmt die Verantwortung für die Aktionen, weshalb der Aktivitätsbereich auch als Verantwortungsbereich bezeichnet wird.
Hinweis: In BPMN heißt dieser Bereich "Pool" und die weitere Unterteilung "Lane" (abgeleitet vom englischen Begriff der "swim lane", dt. Schwimmbahn)

(Aktion) Stellt die fundamentale Einheit ausführbarer Funktionalität dar, die im Modell nicht weiter zerlegt wird und somit atomar ist [:217]. Eine alternative Bezeichnung ist (ausführbarer) Aktivitätsknoten [:216].

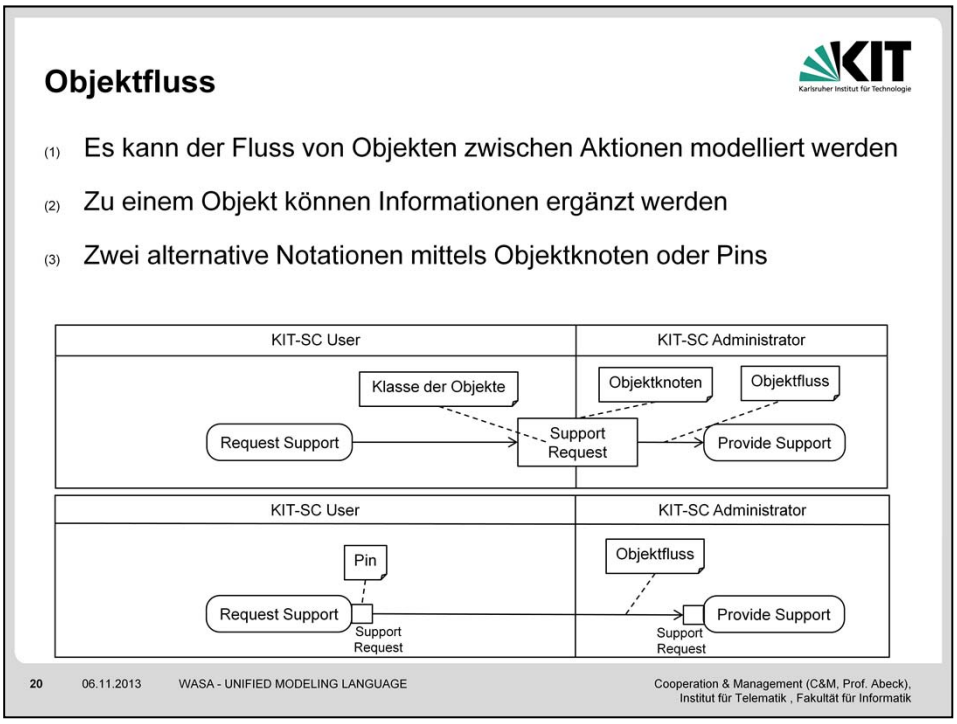
(Pfeile) Die Pfeile mit offener Pfeilspitze geben den Kontrollfluss an und repräsentieren die Ausführungsreihenfolge der Aktivitäten, die sie verbinden [:218].

(Entscheidung) Entscheidungen (engl. decision) werden als Diamanten-geformte Knoten ausgedrückt und ermöglichen die Ausführung unterschiedlicher Aktionsfolgen in Abhängigkeit einer Wächterbedingung (engl. guard condition), die in eckigen Klammern gesetzt werden [MH06:47].

(Gabelung, Vereinigung) Die zwischen der Gabelung (engl. fork) und Vereinigung (engl. join) stehenden Aktionsfolgen werden parallel (engl. concurrently oder in parallel) ausgeführt [MH06:50].

(Signal) Signal-Sendung ist eine spezielle Art einer Aktion, die asynchron ein Signal an ein Zielobjekt sendet, während Signal-Empfang auf den Empfang eines Signals wartet [:234]. Asynchron bedeutet, dass der Sender (im Beispiel der KIT-SC-Nutzer) nach dem Senden des Signals weiterarbeiten kann, d.h. er muss nicht auf die Antwort des Signalempfängers warten.
Die Modellierung dieser Kommunikation zwischen KIT-SC-Nutzer und KIT-SC-Administrator mittels Signal ist aus folgenden Gründen ungünstig:
(i) Die Asynchronität ist in diesem Fall eher nicht gegeben (da der KIT-SC-Nutzer gar nicht sinnvoll weiterarbeiten kann, bevor seine Anfrage bearbeitet wurde, damit er die KIT-SC-Anwendung starten kann).
(ii) Es wäre wünschenswert, die Support-Anfrage näher beschreiben zu können.
Eine alternative Darstellung zu den Signalen stellt der auf der nächsten Seite beschriebene Objektfluss dar.

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.
[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[Ke06:222], [MH06:53]
Datenobjekte stellen häufig einen wichtigen Aspekt des zu modellierenden Prozesses dar.
Hinweis: Objekte müssen nicht zwangsläufig IT-gestützte Objekte sein (z.B. kann es sich auch um ein papierbasiertes Formular handeln) [MH06:53].

(1) Objekte bzw. Objektknoten sind wie Aktionen (= ausführbare Aktivitätsknoten) eine Unterklasse von Aktivitätsknoten (eine dritte Unterklasse sind die Kontrollknoten, wozu Start- und Endeknoten sowie Entscheidungs- und Verbindungsknoten gehören.).

(2) Solche Informationen betreffen den Zustand oder die Anzahl an Objekten, die maximal fließen dürfen.
Im Beispiel:
(i) Zu dem Objekt (Objektknoten bzw. Pin) "Support Request" kann durch Hinzufügen von "[Open]" ausgedrückt werden, dass die Anfragebearbeitung noch offen ist, d.h. die Bearbeitung ist noch nicht abgeschlossen.
(ii) Durch Hinzufügen von "{upperBound=3}" wird ausgedrückt, dass nur maximal 3 Probleme pro Anfrage gemeldet werden dürfen.

(3) Die beiden Notationen sind gleichwertig und betonen nur unterschiedliche Aspekte des Objektflusses.
(i) Bei der Objektknoten-Notation wird das ausgetauschte Objekt betont.
(ii) Die Pin-Notation verdeutlicht, dass ein Objekt der Ausgabeparameter einer Aktion und der Eingabeparameter einer anderen Aktion ist.

Die beiden Notationen werden an einem Ausschnitt des vorherige Beispiels aufgezeigt, bei dem der Support-Wunsch des KIT-SC-Nutzers in Form eines asynchron an den KIT-SC-Administrator gesendetes Signal modelliert wurde.
(Objektknoten) Modelliert die Übergabe von Objekten zwischen Aktionen und kann als eine Art Speicher für Objekte der spezifischen Klasse betrachtet werden [Ke06:222].
(Objektfluss) Repräsentiert den Transport von Objekten.

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.
[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.

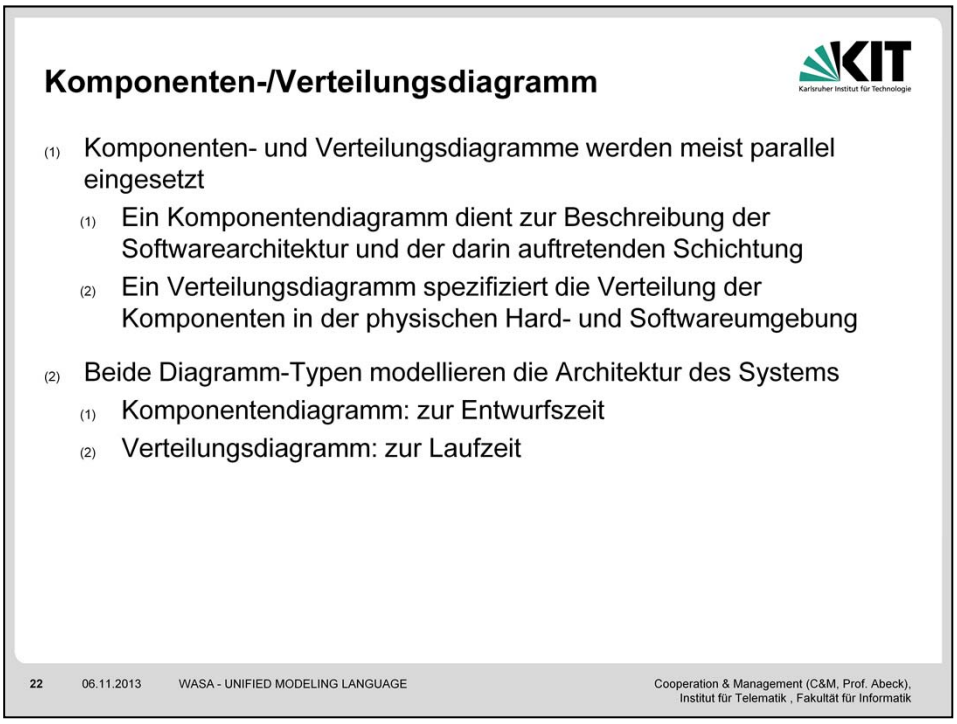
LZ DIAGRAMME – ÜA AKTIVITÄTSDIAGRAMM

(1) Zu welcher Kategorie von Diagrammen gehören die Aktivitätsdiagramme?

(2) In welchen Phasen der Softwareentwicklung lassen sich Aktivitätsdiagramme in welcher Form einsetzen?

(3) Wie werden die an den Abläufen beteiligten Organisationseinheiten bzw. Rollen in Aktivitätsdiagrammen beschrieben?

(4) Welche Formen der Kommunikation zwischen den im Beispiel auftretenden zwei Rollen wurden behandelt?

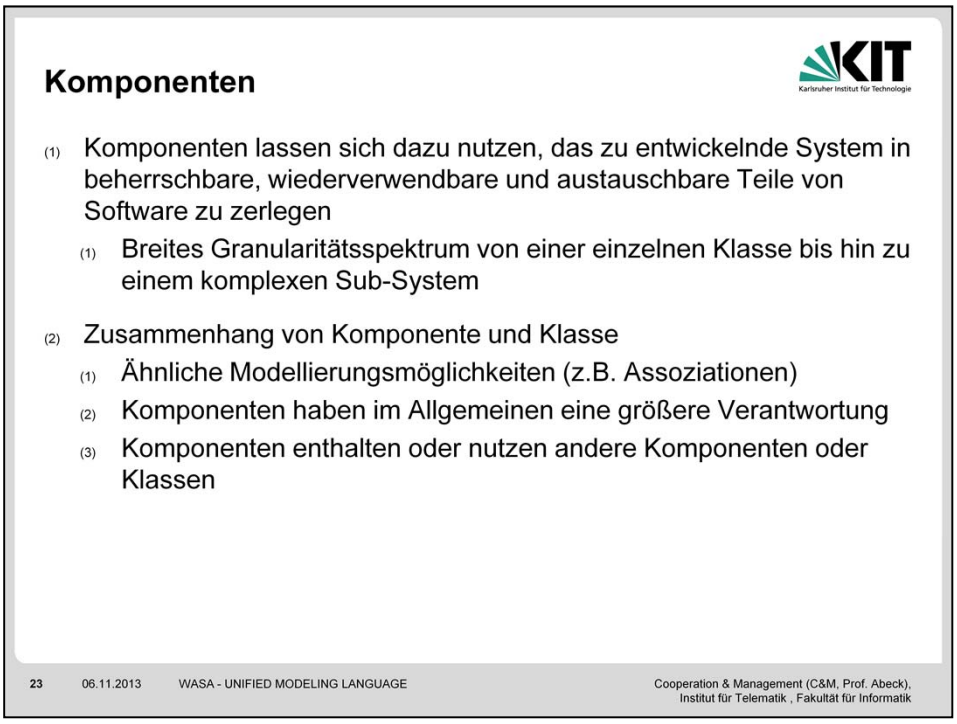


[MH06:186], [Ke06:145, 161]
Das Komponentendiagramm dient zusammen mit dem Verteilungsdiagramm dazu, die Architektur der Anwendung darzustellen. Der Unterschied zwischen den beiden Diagrammen besteht darin, dass das Komponenten-Diagramm die logische Verbindung zwischen den einzelnen Komponenten veranschaulicht und das Verteilungsdiagramm die physische Verteilung dieser Komponenten aufzeigt [Ke06]. Das Verteilungsdiagramm wird üblicherweise nach dem Komponentendiagramm erstellt, da das Wissen über die Komponenten dafür schon bereitstehen sollte.

(1) Beide Diagramm-Typen werden in der Designphase des Softwareentwicklungsprozesses eingesetzt.
(1.1) Eine häufig gewählte Aufteilung der Aufgaben großer Softwaresysteme ist die Drei-Schicht-Architektur (Daten, Geschäftslogik, Präsentation) [Ke06:145].
(1.2) Auf der Grundlage von Verteilungsdiagrammen werden Entscheidungen über eventuell anzuschaffende Hardware- und Softwarekomponenten und deren Kommunikationswege getroffen [Ke06:161].

(2.1) Die Softwarearchitektur und damit das Komponentendiagramm sind Teil der Entwicklungssicht auf das Modell des Systems [MH06:186].

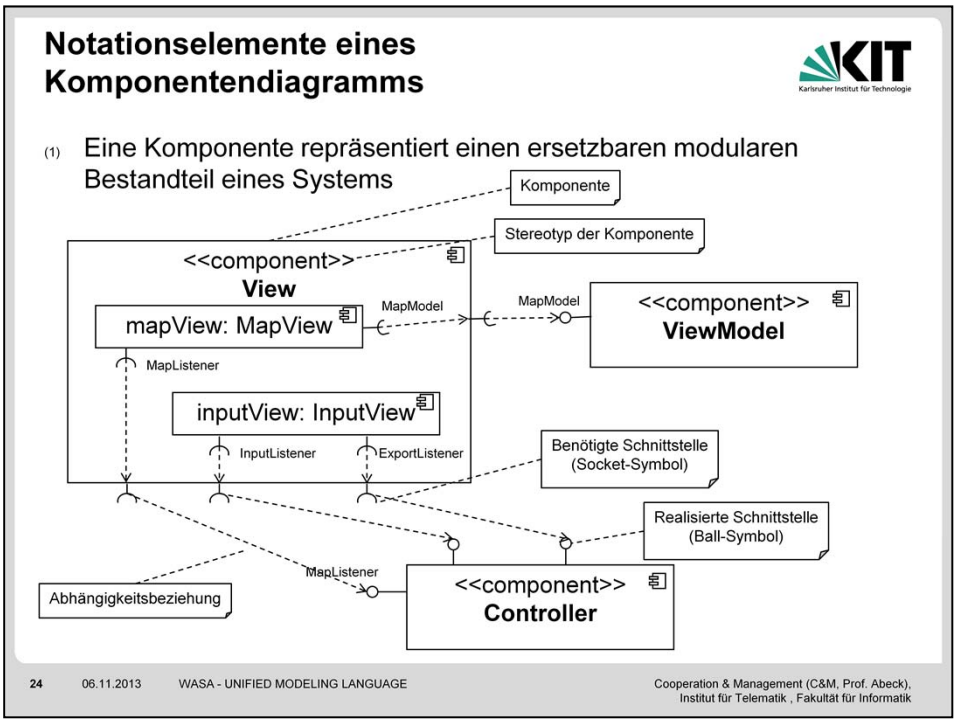
[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.
[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[MH06:187]
(1) Die Komponenten liegen in gewisser Weise zwischen den Anforderungen und den Klassen, da diese Lücke bei komplexeren Systemen groß ist und daher nicht unmittelbar Klassen aus den Anforderungen abgeleitet werden können.
(2.1) Auf die Ähnlichkeiten und Unterschiede von Komponenten und Klassen wird im nächsten Punkt noch näher eingegangen.

(2.1) Weitere Möglichkeiten, die sowohl Komponenten und Klassen bieten, sind [:187]:
- Generalisierung (das ist z.B. ein wesentlicher Unterschied zwischen Komponenten und Services, bei denen aufgrund der losen Kopplung keine Vererbung unterstützt wird).
- Implementierung von Interfaces
- Besitz von Operationen
(2.2) Beispiel: Eine Klasse "User Account", die Informationen zu Benutzer enthält und eine Komponente "User Management", die die Erstellung und Überprüfung (Authentifizierung) von User Accounts ermöglicht.
(2.3) Dadurch werden die Komponenten im Vergleich zu Klassen zu größeren Funktionseinheiten mit größerer Verantwortung.
Ein wesentlicher Unterschied zwischen Komponenten und Klassen (an dem auch deutlich wird, dass Komponenten "umfassender" sind als Kassen) besteht darin, dass Komponenten Klassen beinhalten können, aber Klassen keine Komponenten.

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.
[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[Ke06:145]
Ein Komponentendiagramm ist (wie z.B. auch ein Klassendiagramm) ein Strukturdiagramm. Es eignet sich besonders gut zur Spezifikation von Softwarearchitekturen.

(1) Durch die Komponente wird das Innere dieses System-Bestandteils gekapselt.

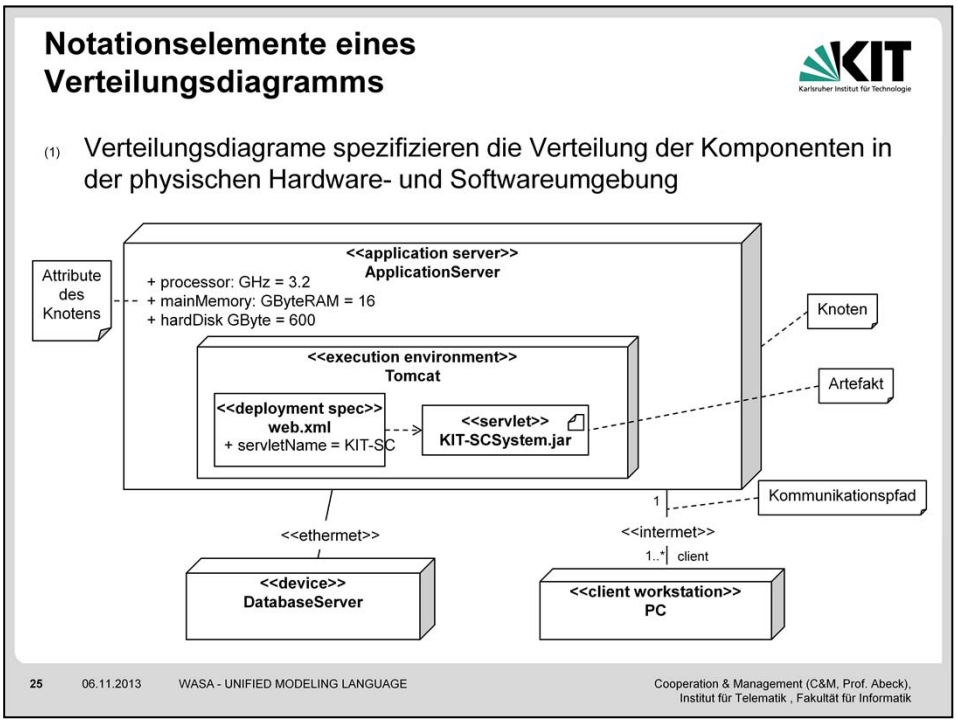
Das beispielhafte Komponentendiagramm beschreibt einen Ausschnitt aus der in [CM-PR-PSE1011-E-T1:2.1] beschriebenen Softwarearchitektur des KIT-Smart-Campus.
(View, ViewModel, Controller) Anwendung des Entwurfsprinzips Model-View-Controller (MVC). Der Begriff des ViewModel deutet an, dass das Modell ausschließlich die Präsentationsinhalte enthält und nicht die Datenhaltungsschicht realisiert. Das MVC-Prinzip dient hier zur Strukturierung der Präsentationsschicht (und nicht der gesamten Architektur der KIT-SC-Anwednung).
(<<component>> View) Beispiel einer Komponente, wobei <<component>> der Stereotyp der Komponente und "View" der Name der Komponente ist [Ke06:147].
Die Komponente "View" realisiert die graphische Darstellung der KIT-SC-Anwendung im Browser und leitet Aktionen und Ereignisse zur Interpretation an den Controller weiter.

(Benötigte/Realisierte Schnittstelle) Durch die benötigten (engl. required) Schnittstellen (-> Socket-Symbol) und die bereitgestellten (engl. provided) Schnittstellen (-> Ball-Symbol) wird das Verhalten einer Komponente vollständig definiert.
(Abhängigkeitsbeziehung) Diese durch einen gestrichelte Linie mit offener Pfeilspitze dargestellte Beziehung beschreibt eine Abhängigkeit und drückt eine Kunden-Dienstleister-Beziehung (engl. client-supplier) aus [Ke06:72].
Hier wird die Beziehung nur zur vereinfachten Darstellung der beiden Schnittstellen-Symbole genutzt. Eine alternative Darstellung besteht darin, keine Abhängigkeitsbeziehung zwischen dem Socket-Symbol und dem Ball-Symbol vorzusehen (kombiniertes Ball-Socket-Symbol) [MH06:191].

(MapModel) Schnittstelle der Unterkomponente "MapView", durch die für das Zeichnen benötigten Informationen von der Komponente "ViewModel" erfragt werden.

MVC Model-View-Controller

[CM-PSE-WS10/11] Cooperation & Management: Praxis der Software-Entwicklung (PSE), WiSe10/11, Entwurfsdokument (E), Team1 (T1), Karlsruher Institut für Technologie (KIT).
[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.
[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[Ke06:161]
(1) Verteilungsdiagramme werden in der Entwurfsphase parallel zu den Komponentendiagrammen erstellt. Sie dienen dazu, Entscheidungen über eventuell anzuschaffende HW-/SW-Komponenten zu treffen und die Laufzeitumgebung während der Implementierung des modellierten Softwaresystems aufzubauen.

(Knoten) Ein Knoten (engl. node) repräsentiert eine Systemressource.

(device, application server, execution environment, ...) Arten von Knoten, die in UML unterschieden werden. So ist <<device>> der allgemeinste Knotentyp, durch den eine physische Recheneinheit repräsentiert wird.

(Attribute des Knotens) Zu einem Attribut werden drei Angaben gemacht: Attributname ("processor"), Attributtyp ("Ghz"), Vorgabewert ("3.2")

(Kommunikationspfad) Definiert eine Assoziation zwischen zwei Knoten, über die Signale und Nachrichten ausgetauscht werden können [:167].
Wie zu jeder Assoziation, können einem Kommunikationspfad Attribute wie Multiplizität ("1..*"), Rollen ("client") oder Stereotypen ("<<internet>>") zugeordnet werden.

(Artefakt) Modellieren physische Informationseinheiten wie Dateien, Datenbanken, Datenbanktabellen [:160].

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.

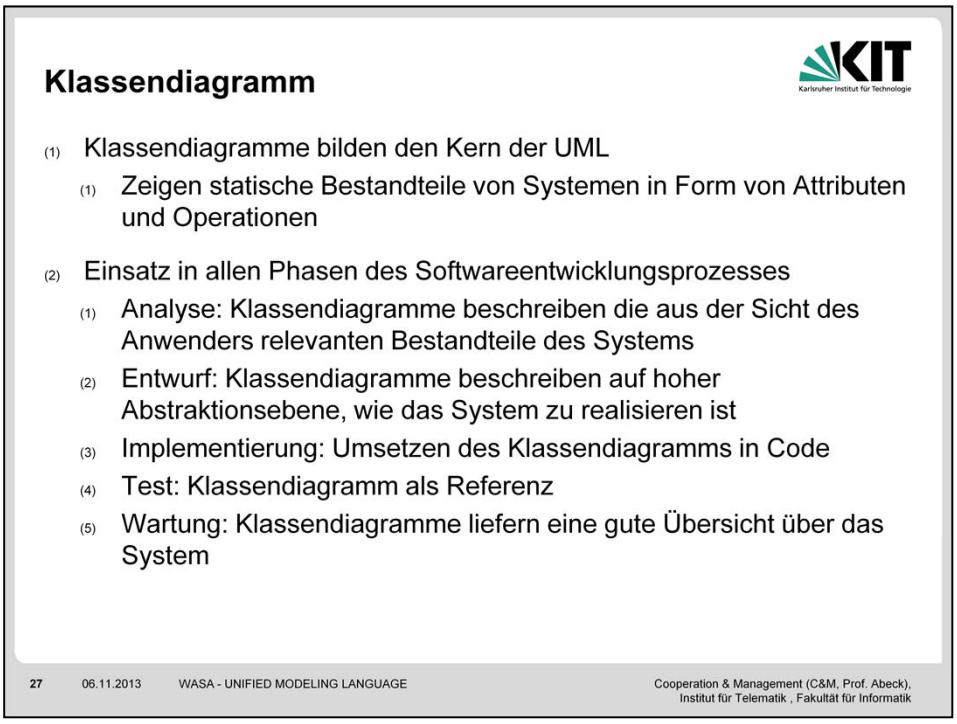
LZ DIAGRAMME – ÜA KOMPONENTEN-/VERTEILUNGSDIAGRAMM

(1) Welche Gemeinsamkeiten weisen ein Komponentendiagramm und ein Verteilungsdiagramm auf?

(2) Welche Eigenschaften sollte eine Komponente aufweisen?

(3) Was ist der wesentliche Unterschied zwischen einer Komponente und einer Klasse?

(4) Durch welche Notationselemente wird die Schnittstelle einer Komponente beschrieben?

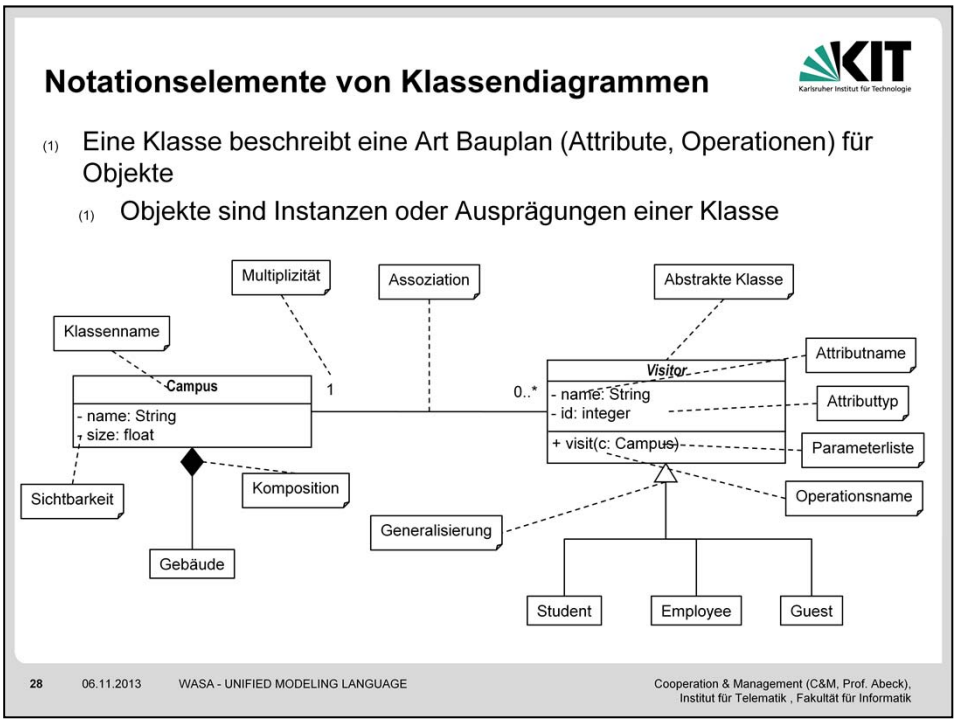


[Ke06:29]
Als ein weiteres Beispiel eines Strukturdiagramms wird das Klassendiagramm näher behandelt.

(1) Klassendiagramme sind aus der heutigen objektorientierten Softwareentwicklung nicht mehr wegzudenken.
(1.1) Dynamische Aspekte der Attribute und Operationen werden nicht in Klassendiagrammen behandelt.

(2) Besonders intensiv werden Klassendiagramme in den ersten beiden Phasen, der Analyse und dem Entwurf, genutzt [:29].
(2.1) In der Analysephase repräsentieren die Klassen Geschäftsobjekte und die Klassendiagramme sind bewusst einfach gehalten, um mit dem Anwender bzw. Auftraggeber mittels dieser Diagramme kommunizieren zu können.
(2.2) Im Entwurf werden so genannte logische Klassendiagramme verwendet, durch die interne Strukturen des Systems aufgezeigt werden. Diese sind viel detailreicher als die in der Analysephase aufgestellten Klassendiagramme.
(2.3) Umsetzung der im Entwurf erstellten Klassen in Programmcode der jeweiligen Zielprogrammiersprache. Das Kassendiagramm unterstützt die Kommunikation zwischen Softwarearchitekten und Programmierer [:30]
In der modellgetriebenen Entwicklung ist das Ziel, die Umsetzung automatisch durch eine Transformation durchzuführen.
(2.4) Prüfung der Implementierung auf Vollständigkeit und Korrektheit gegen das Klassendiagramm.
(2.5) Dadurch wird die Gefahr gemindert, sich bei der Durchführung von Wartungsarbeiten (z.B. Fehlersuche) im Code zu verlieren.

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



[Ke06:32]
Ein Klassendiagramm gehört zu der Art der UML-Strukturdiagramme.

(1) Die Attribute beschreiben die Struktur und die Operationen das Verhalten des Bauplans.
(1.1) 'Instanziierung' bezeichnet die Erstellung eines Objekts nach dem Bauplan der Klasse.

(Klassenname) Sollte mit einem Großbuchstaben beginnen und ausschließlich Buchstaben (keine Umlaute) und Zahlen enthalten [:33].

(Assoziation) Ist binär und spezifiziert eine semantische Beziehung zwischen zwei Klassen, die miteinander interagieren (d.h. Operationen aufrufen) können [:49].
Es lässt sich in UML auch eine n-äre Assoziation, also ein Beziehung zwischen n Klassen, in Form einer Raute modellieren. Da n-äre Assoziationen z.B. in Programmiersprachen wie Java nicht direkt unterstützt werden, muss eine zusätzliche Klasse eingeführt werden, die jeweils binäre Assoziationen mit den beteiligten n Klassen aufweist [:58].

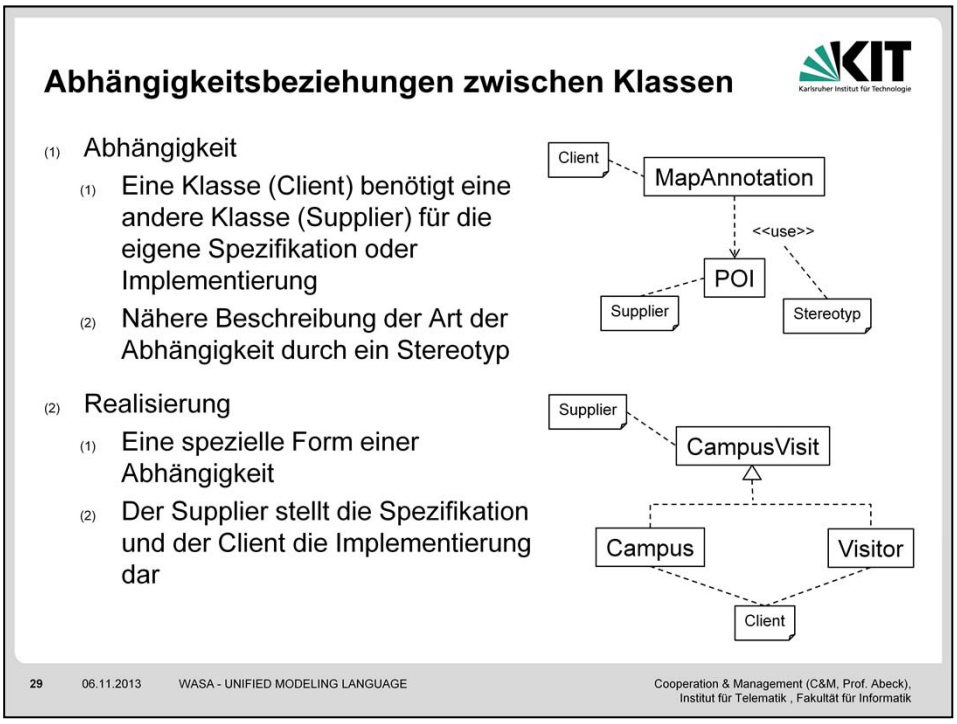
(Multiplizität) Gibt die Anzahl der Instanzen an, die in der Assoziation auftreten dürfen [:50].
Fehlt die Angabe der Multiplizität, so ist diese 1 [:35].
Im Beispiel:
- Ein Besucher besucht genau einen Campus (d.h., eine Instanz oder ein Objekt der Klasse "Campus").
- Ein Campus kann von beliebig vielen ("0..*", alternative Schreibweise: "*") Besuchern besucht werden.

(Generalisierung) Modelliert eine Beziehung zwischen einer spezifischen Subklasse und einer allgemeinen Superklasse; definiert eines der zentralen Konzepte der Objektorientierung [:75].

(Abstrakte Klasse) Ein ggf. unvollständiger Bauplan, so dass hieraus keine Objekte direkt instanziiert werden können (sondern nur aus den aus der abstrakten Klasse abgeleiteten Klassen) [:88].

(Komposition) Starke Form der Aggregation (Ganzes-Teile-Beziehung, spezielle binäre Assoziation), bei der die Verbindung zwischen den Teilen und dem Ganzen als untrennbar definiert wird. Das bedeutet, dass die Existenz (Lebensdauer) des Teils direkt von der Existenz des Ganzen abhängt.
Im Beispiel: Ein Raum kann ohne das Gebäude, zu dem es gehört, nicht existieren: Wird das Gebäude abgerissen (Ende der Lebensdauer des Ganzen), so existiert auch nicht mehr der Raum (= Teil des Ganzen).

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



[Ke06:72]
(1.1) Eine Abhängigkeit wird auch als Client-Supplier-Beziehung (dt. Kunde-Dienstleister-Beziehung) bezeichnet und drückt aus, dass der Client ohne den Supplier nicht vollständig ist und ohne ihn nicht ausgeführt werden kann. Daher wirken sich Änderungen beim Supplier auch immer auf den Client aus.
(1.2) Beispiele für Stereotypen, durch die die Art der Abhängigkeit beschrieben wird [:73]:
<<call>>: Der Client ruft mindestens eine Operation des Supplier auf.
<<permit>>: Der Supplier gewährt dem Client Zugriff auf seine (privaten) Attribute und Operationen.
<<use>>: Die Existenz des Supplier wird für die Funktionsfähigkeit des Client benötigt.

(MapAnnotation <<use>> POI) Das Beispiel drückt aus, dass die Existenz des Supplier (hier: POI, der PointOfInterest) für die Funktionsfähigkeit des Client (hier: MapAnnotation, ein Hinweis auf der Karte) benötigt wird.

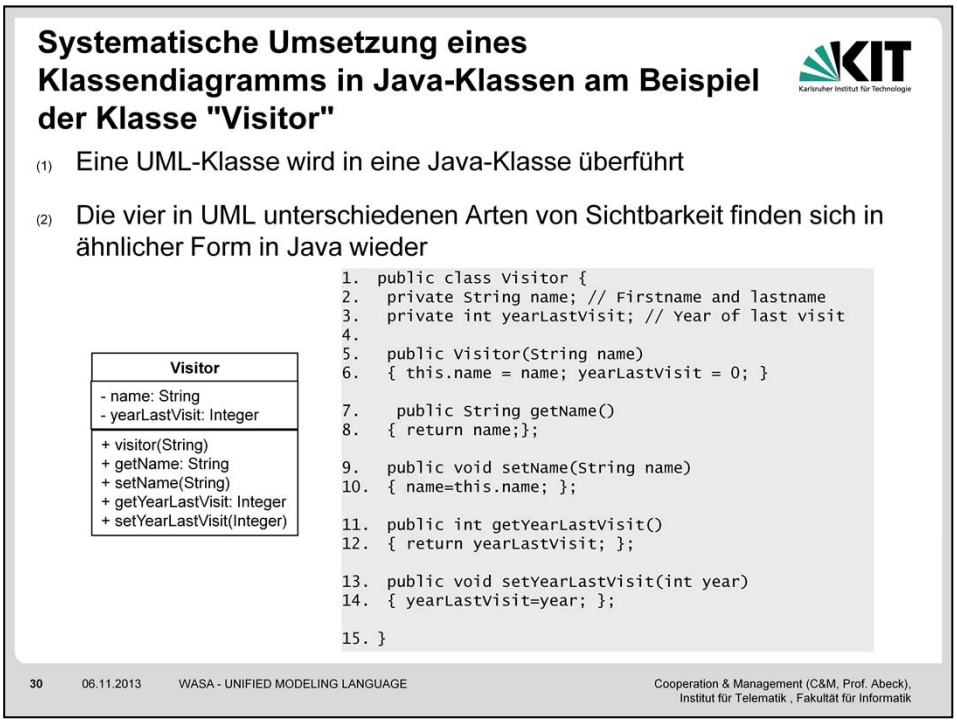
(Stereotyp) <<use>> ist ein Beispiel eines Stereotyp, was an den spitzen Klammern ersichtlich ist. Durch ein Stereotyp wird allgemein die Art eines Notationselements spezifiziert [:85] und es gibt eine Auskunft über den Zweck und die Rolle dieses stereotypisierten Elements.

(2.1) Es bestehen zwei Unterschiede hinsichtlich der Notation der Realisierungsbeziehung im Vergleich zur Abhängigkeitsbeziehung:
(i) geschlossene Pfeilspitze
(ii) kein expliziter Stereotyp

(CampusVisit) Wird realisiert durch Campus und Besucher.

POI PointOfInterest

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



[MH06:68]

(1) Die Kassenkonzepte von UML stimmen mit dem Klassenkonzept, das einer objektorientierten Sprache wie z.B. Java zugrunde liegt, im Wesentlichen überein. Daher lassen sich die in der Beschreibung einer UML-Klasse auftretenden Elemente weitestgehend 1:1 auf entsprechende Java-Sprachelemente transformieren. Die ersten Transformationsregeln können informell so formuliert werden:
(i) Das UML-Klassensymbol (Rechteck) wird in das Java-Schlüsselwort "class" transformiert.
(ii) Der UML-Klassenname wird als Java-Klassenname übernommen.
(iii) UML-Klassenattribute werden zu Java-Datenfeldern in der Klassendeklaration.
(iv) UML-Klassenoperationen werden zu Java-Methoden.

(2) Im vorliegenden Fall sind alle Attribute privat und alle Operationen sind öffentlich. Dies ist die empfohlene Sichtbarkeitsfestlegung, die dem Geheimnisprinzip entspricht (der durch die Attribute festgelegte interne Zustand eines Objekts ist aufgrund der "privaten" Sichtbarkeit von außen nicht zugreifbar).

In UML werden vier Typen von Sichtbarkeit (engl. visibility) unterschieden (angefangen vom umfassendsten Zugriff):
(i) "+": Public - Zugriff durch jede beliebige Klasse
(ii) "#": Protected – Zugriff innerhalb der Klasse und allen Unterklassen
(iii) "~": Package – Zugriff durch alle Klassen, die sich im selben Paket wie die Klasse befinden
(iv) "-": Private – Zugriff nur innerhalb der Klasse

(get*, set*) Diese Methoden, durch die ein privates Attribut gelesen (geholt, engl. get) und geschrieben (gesetzt, engl. set) wird, werden auch als "getter- und setter-Methoden" bezeichnet.

[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.

LZ DIAGRAMME – ÜA KLASSENDIAGRAMM

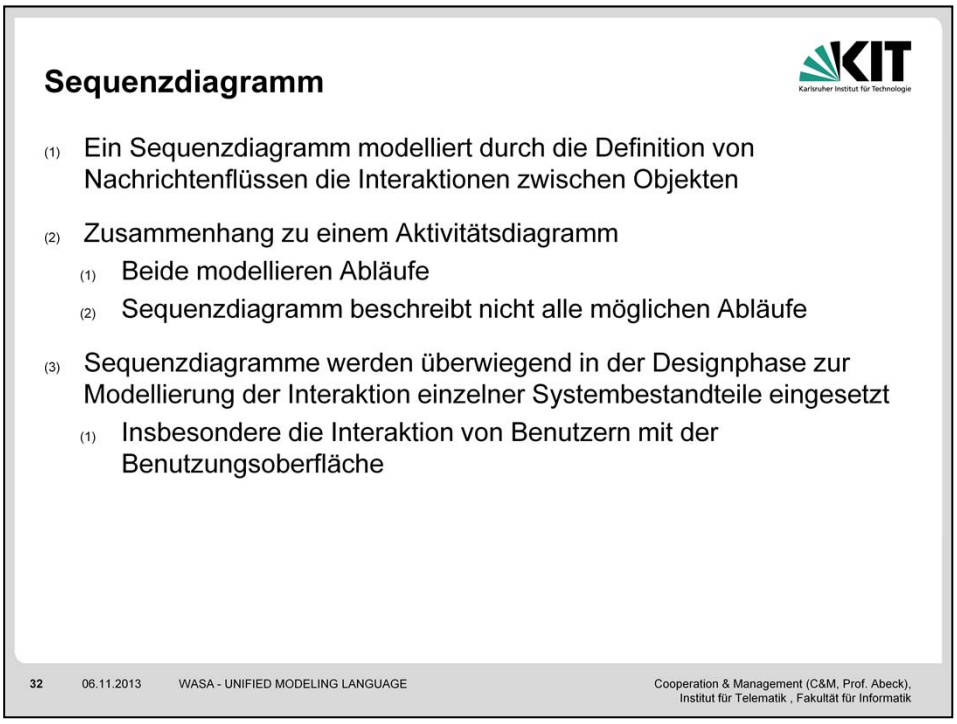
(1) Was wird mittels Klassen beschrieben
  (1) in der Analysephase?
  (2) in der Entwurfsphase?

(2) Wie können Klassendiagramme die Wartungsarbeiten an einem Softwaresystem unterstützen?

(3) Aus welchen (drei) Bestandteilen setzt sich eine Klasse zusammen?

(4) Was ist eine Komposition?

(5) Welche Rollen bestehen in einer Abhängigkeitsbeziehung bzw. einer Realisierungsbeziehung?

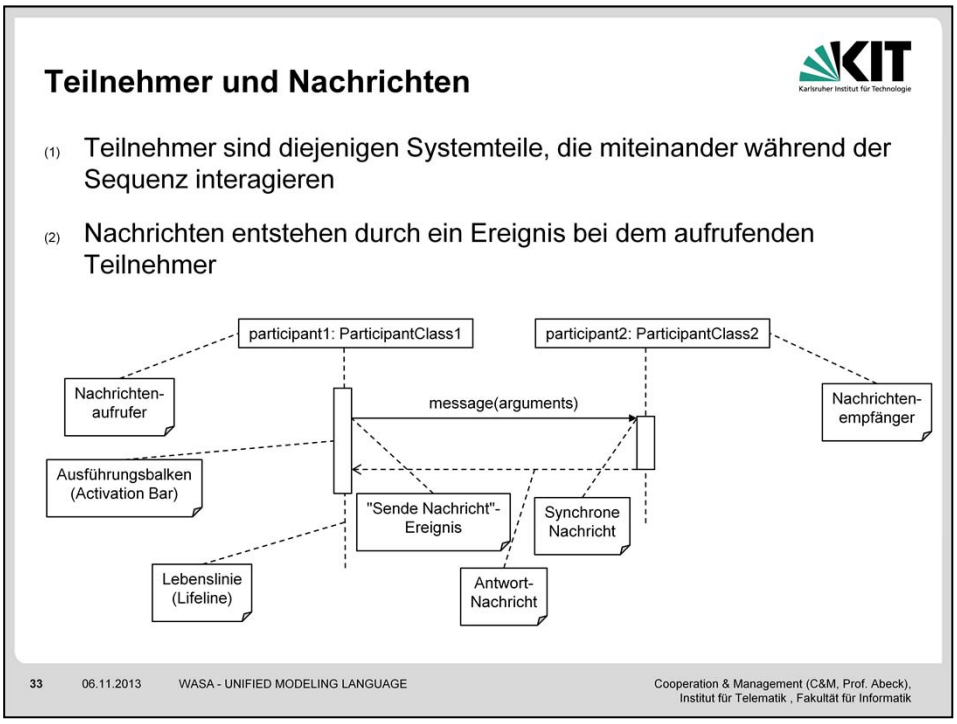


[Ke06:339]
(1) Sequenzdiagramme beinhalten aber keine Informationen über Beziehungen der Objekte.

(2) Die beiden Diagramm-Typen gehören neben dem Anwendungsfalldiagramm zu den am meisten verwendeten Verhaltensdiagramm-Typen [Ke06:21].
(2.1) Die Abläufe beschreiben relevante dynamische Aspekte des Systems.
(2.2) Eine Sequenz beschreibt immer genau einen Ablauf von verschiedenen möglichen Abläufen, die im System auftreten können.

(3) Die Systembestandteile sind üblicherweise Komponenten oder Subsysteme.
Prinzipiell ist ein Einsatz auch in den übrigen Phasen denkbar:
- Analyse: In Geschäftsprozessen zwischen den Geschäftspartnern auftretende Nachrichtenflüsse
- Implementierung: Verwendung der in der Analyse-/Designphase erstellten Sequenzdiagramme als Realisierungsvorlage [:340].
- Test: Modellierung und Dokumentation von Testfällen
(3.1) Diese dynamischen Aspekte einer Anwendung entsprechen der Präsentationslogik und sind demgemäß Teil der Präsentationsschicht der Anwendungsarchitektur.

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.



[MH06:108]
(1) Schwerpunkt des Sequenzdiagramms ist die Erfassung der Reihenfolge (also der Sequenz) von Interaktionen zwischen den Teilnehmern (engl. participants) [:110].

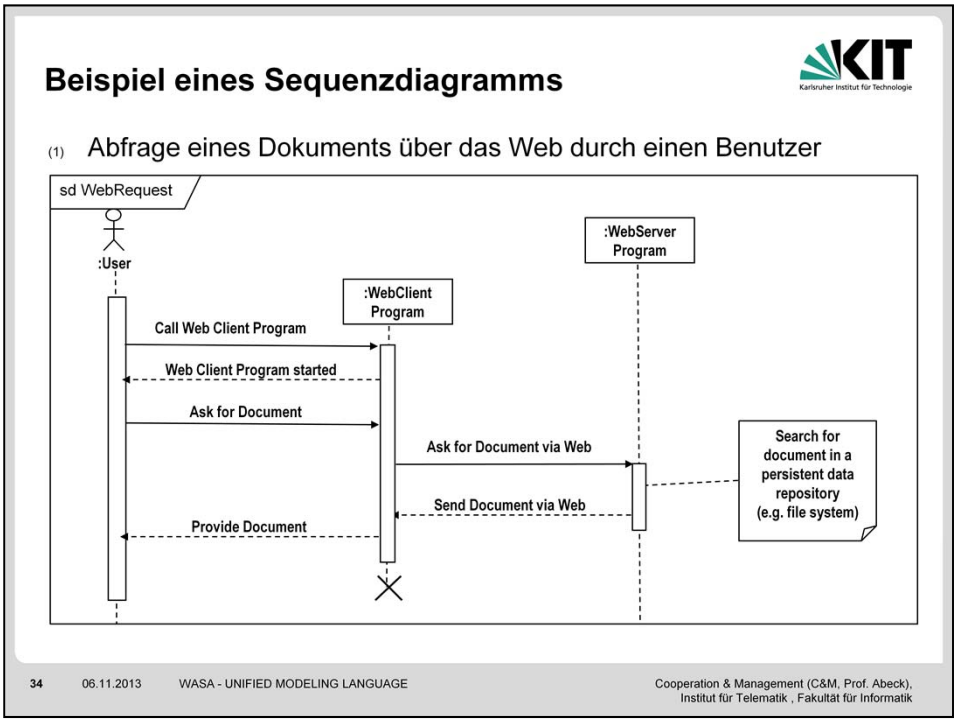
(2) Ein Ereignis (engl. event) ist der kleinste Teil einer Interaktion und beschreibt einen beliebigen Punkt in der Interaktion, in der etwas passiert [:111].
Ereignisse stehen in einem direkten Zusammenhang zu Nachrichten (engl. message), da das Senden und Empfangen jeweils einem Ereignis auf Sender- und Empfänger-Seite entsprechen.
Hinweis: Neben Nachricht existiert der in diesem Kontext als gleichwertig anzusehende Begriff des Signals: Während Nachricht bevorzugt von einem Software-Designer verwendet wird, spricht der System-Designer von einem Signal.

(participant1: ParticipantClass1) "participant1" ist der Name des Systemteils und "ParticipantClass1" ist die Klasse, zu der der Systemteil gehört.
Es können zahlreiche weitere Formate zur Benennung von Teilnehmern gewählt werden. Zu beachten ist, dass keine Vorgaben hinsichtlich Klein-/Großschreibung bestehen (das gilt grundsätzlich nicht nur für Teilnehmer, sondern auch z.B. für Klassen – d.h., die z.B. auch in Java übliche Konvention, Klassennamen groß zu schreiben, wird von UML nicht zwingend vorgeschrieben).

(message(arguments)) Das allgemeine Schema zur Beschreibung der Nachrichtensignatur lautet [:112]:
attribute = signal_or_message_name (argname1: argclass1, argname2: ...) : return_type

(Synchrone Nachricht) Die synchrone Kommunikation wird durch die geschlossene Pfeilspitze ausgedrückt und besagt, dass der Nachrichtenaufrufer nach dem Senden der Nachricht auf den Empfang der Nachricht wartet (d.h. bis zu diesem Zeitpunkt blockiert ist).
Die synchrone Kommunikation entspricht genau einem regulären Java-Methodenaufruf. Eine asynchrone Kommunikation (in UML durch eine offene Pfeilspitze ausgedrückt) entspricht in Java der Nutzung des Thread-Konzepts, um parallel zum Bearbeitungsprozess auf den Empfang zu warten und damit die Weiterarbeit nicht zu blockieren.

[MH06] Russ Miles, Kim Hamilton: Learning UML 2.0, O'Reilly Media, 2006.



[Ke06:339]
(1) Das Sequenzdiagramm zeigt, wie ein Benutzer und die zwei beteiligten Systeme miteinander interagieren [:339]. Details zu diesem Ablauf, insbesondere zu den genutzten Kommunikationsprotokollen, erfolgt in der WASA-Kurseinheit KOMMUNIKATION.

(User, WebClientProgram, WebServerProgram) Jeder Teilnehmer (also beteiligte Person oder beteiligtes System) ist durch jeweils eine Lebenslinie repräsentiert. Ist die Lebenslinie gestrichelt, so bedeutet das, dass der Teilnehmer gerade passiv ist. Der Übergang in den aktiven Zustand wird durch den sog. Ausführungsbalken ausgedrückt.

Die Notation der Bezeichnung eines Teilnehmers lautet: <Name>:<Typ>. Die Angabe des Namen ist optional, wie das Beispiel an den beteiligten Systemen Web-Client-Programm und Web-Server-Programm (beides Typen) zeigt.
Die Interaktion zwischen den Teilnehmern wird durch Austausch von Nachrichten beschrieben. Eine ausgefüllte Pfeilspitze heißt, dass die Nachricht synchron versendet wird, d.h. der Sender wartet auf die Ausführung der mittels der Nachricht aufgerufenen Operation. Soll die Interaktion asynchron ablaufen, d.h. der Sender arbeitet nach dem Senden der Nachricht ohne Warten weiter, ist eine offenen Pfeilspitze zu verwenden [:347].

[Ke06] Christoph Kecher: UML 2.0 – Das umfassende Handbuch, Galileo Press, 2006.

LZ DIAGRAMME – ÜA SEQENZDIAGRAMM

(1) Was sind die wesentlichen Unterschiede von Sequenzdiagrammen und Aktivitätsdiagrammen?

(2) Was ist ein typischer Einsatzbereich von Sequenzdiagrammen?

(3) Wie ist der Zusammenhang zwischen Ereignissen und Nachrichten?

(4) Wie heißt die Kommunikationsform, bei der ein Teilnehmer nach dem Senden einer Nachricht nicht bis zum Empfang einer Antwort warten muss? Was ist ein typisches Beispiel für diese Form der Kommunikation?