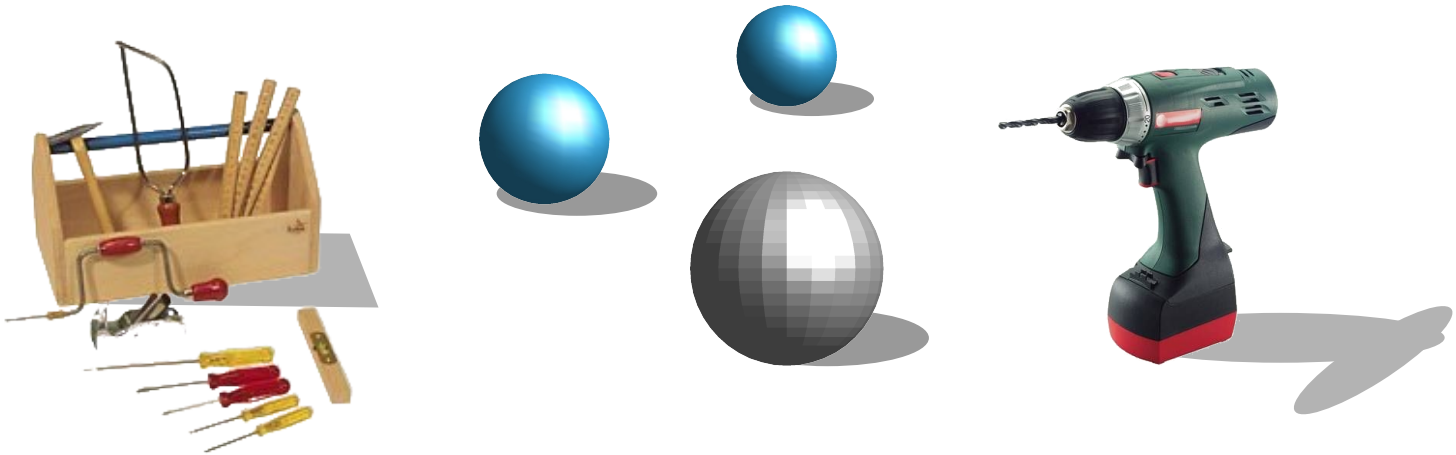
Agentenorientierte Softwareentwicklung: AOSE



20.06.2007

Und nun... ?

- Erstellung einzelner Agenten so weit geklärt
- Anwendungen gefunden
- Offene Fragen
  - Was soll mit Agenten, was ohne realisiert werden?
  - Welche Rollen spielen die Agenten?
  - Wie interagieren sie?
  - Kann man irgendwie **systematisch** vorgehen?

Gefahren beim Anwendungsentwurf

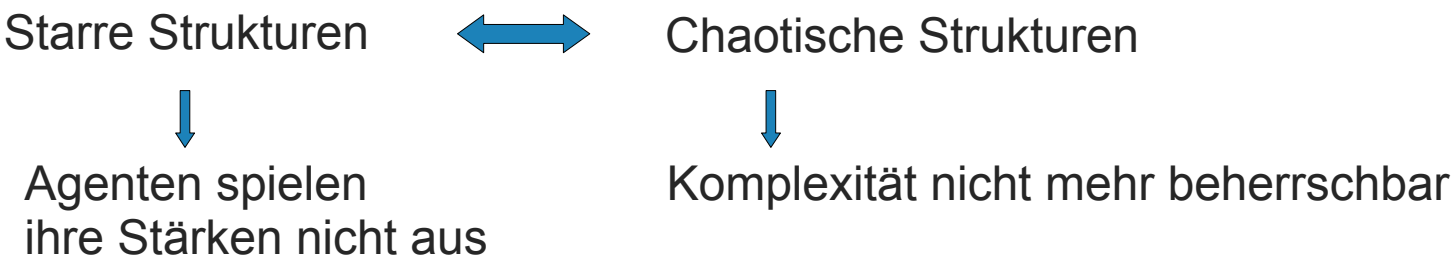
- Überschätzung: „Agenten sind die Lösung auf alle Fragen.“
  - Agenten sind letztlich auch nur Software
  - unterliegen den gleichen Problemen
  - hängen von Fortschritten in den verwendeten Technologien ab
- Dogmatismus: „Agenten sind einfach besser.“
  - Niemals die Alternativen vergessen.
- Ziellosigkeit: „Wir nehmen Agenten, weil sie ein viel versprechender, neuer Ansatz sind.“
  - Nicht auf den Marketingzug aufspringen

Gefahren

- Überzüchtung: Die eierlegende Wollmilchsau für ein einfaches Problem anstreben
- Komplexität: Agentensysteme sind stets Multi-Thread-Anwendungen
- Vergebene Chancen: Die Parallelität nicht nutzen
- Falsche Einteilung
  - Zu viele Agenten: Nicht alles muss ein eigener Agent sein
  - Zu wenige Agenten: Den Multiagentenansatz nicht verfolgen

Gefahren

- Das Rad neu erfinden
  - Ein eigenes Agentensystem bauen, um damit das Problem zu lösen
  - Zu viel Arbeit für infrastrukturelle Fragen aufwenden
- Falsche Organisaton



Vorgehen

- Entsprechung in der traditionellen Welt
  - Modellierung (modellbasierter Entwurf)
  - Entwurfsmuster (Design patterns)
- Ziel in zwei Stufen
  - Agenten als Bausteine zum Aufbau komplexer Anwendungen
  - Gestaltung der einzelnen Agenten
- Wichtig
  - Die Anwendung als Gemeinschaftsaufgabe der Agenten begreifen

Bisherige Ansätze

- Ansätze der strukturierten Entwicklung
  - UML: Modellierung
  - objektorientierte Entwicklung
  - komponentenbasierte Entwicklung
  - Entwurfsmuster (patterns)
  - Warum neue Ansätze?
- Problem
  - Bisherige Ansätze wenden sich an statische Entwürfe
  - agententypische Eigenschaften finden keinen Eingang in den Entwurf

Vorgestellte Ansätze

- Gaia (Wooldridge u.a.)
  - Systematische Vorgehensweise zur Analyse eines Systems
  - Erstellung eines Entwurfsmodells
  - Danach problemspezifische Implementierung
- AgentUML (Bauer, Müller, Odell)
  - Erweiterung von UML 1.4

Gaia

- Methodologie von Wooldridge und Jennings
- Behandelt zwei Ebenen
  - Makroebene (Gemeinschaft) und
  - Mikroebene (Individuum)
- Grundlage
  - Entwurf der Organisation eines Systems
  - System beinhaltet Entitäten, die Rollen einnehmen
  - Rollen liefern Grundlage für Interaktionen

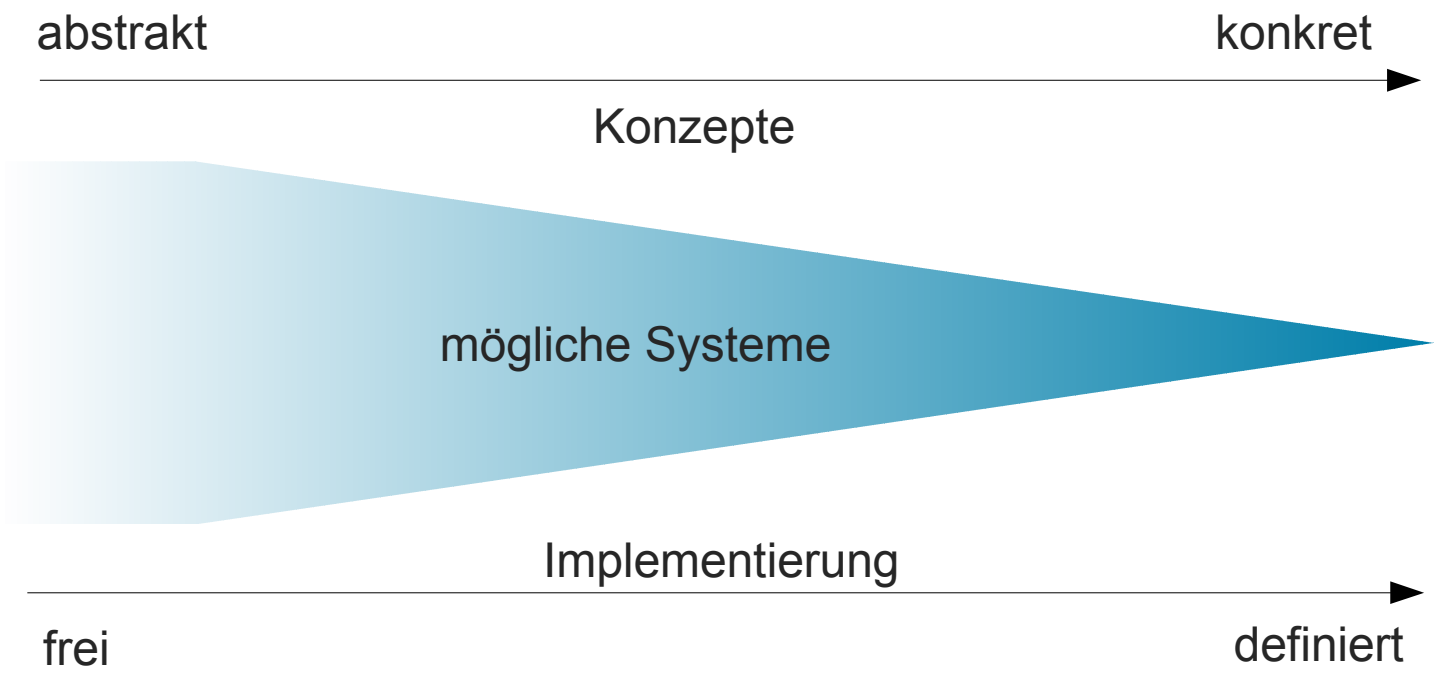
Gaia und bisherige Ansätze

- Bisherige Ansätze sind für die Modellierung nicht unbedingt geeignet
  - Objekte haben ein zu starres Verhalten
  - Problemlösungsstrategien von Agenten erfordern Flexibilität
  - Reichhaltigkeit der Agenteninteraktionen
  - Organisationsstrukturen von Agentengemeinschaften können sehr komplex werden
- Folgen der Anwendung bisheriger Ansätze
  - Vorteile der Agentenanwendungen werden nicht herausgearbeitet, sondern von alten Mustern dominiert
  - Es kommt eine traditionelle Struktur heraus, nur mit Agenten

Gaia: Voraussetzungen

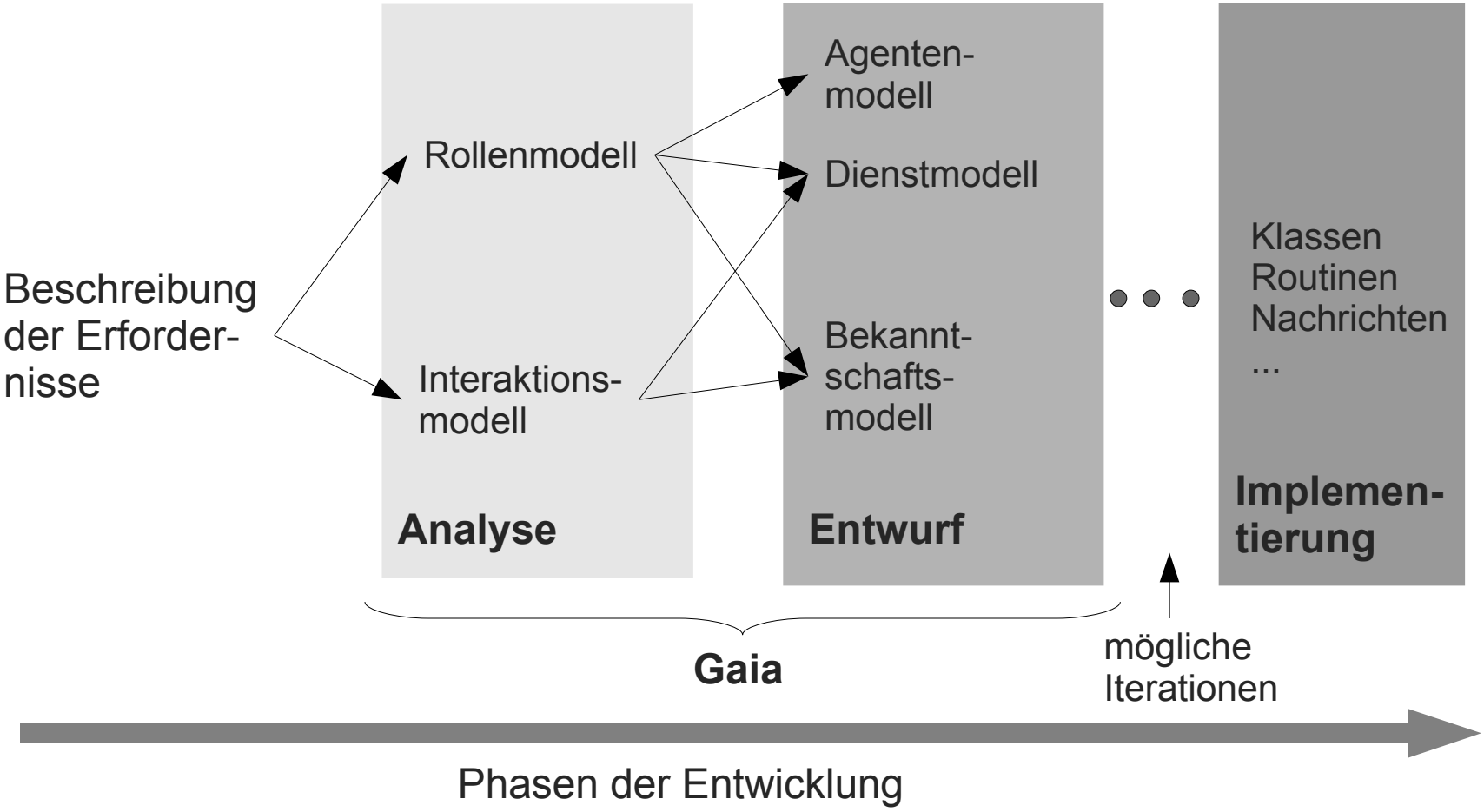
- Agenten sind grobgranulare Verarbeitungseinheiten, die Ressourcen verbrauchen (nutzen Prozesse oder Threads)
- Das Gesamtsystem strebt einen optimalen Zustand an
  - Die Behandlung schwerer Konflikte ist in Gaia nicht vorgesehen
- Keine bestimmte Plattform
  - Agenten sind heterogen: keine Vorgaben an die Programmiersprache, an das Zielsystem, an die Technik
- Statische Organisationsstruktur zur Laufzeit
  - feste Beziehungen zwischen Agenten
  - festgelegte Fähigkeiten
- Überschaubare Anzahl an Agententypen (<100)

Vorgehensweise



Man lässt sich zu Beginn des Prozesses maximale Entscheidungs-freiheit bezüglich der Realisierung; erst am Ende spitzt sich der Prozess zur Auswahl einer Zielplattform zu.

Gaia: Konzeptionelles Rahmenwerk



Konzepte

- Abstrakte Konzepte
  - benötigt in der Analysephase
  - haben *keine unmittelbare* Auswirkung auf die Realisierung

- Rolle
- Berechtigung
- Verantwortung
- Fortschritt
- Funktions-sicherheit
- Protokoll
- Aktivität

- Konkrete Konzepte
  - finden in der Entwurfsphase Verwendung
  - Die Implementierung weist den Konzepten entsprechende Gegenstücke zu

- Agententyp
- Dienst
- Bekanntschaft

Rolle als Konzept

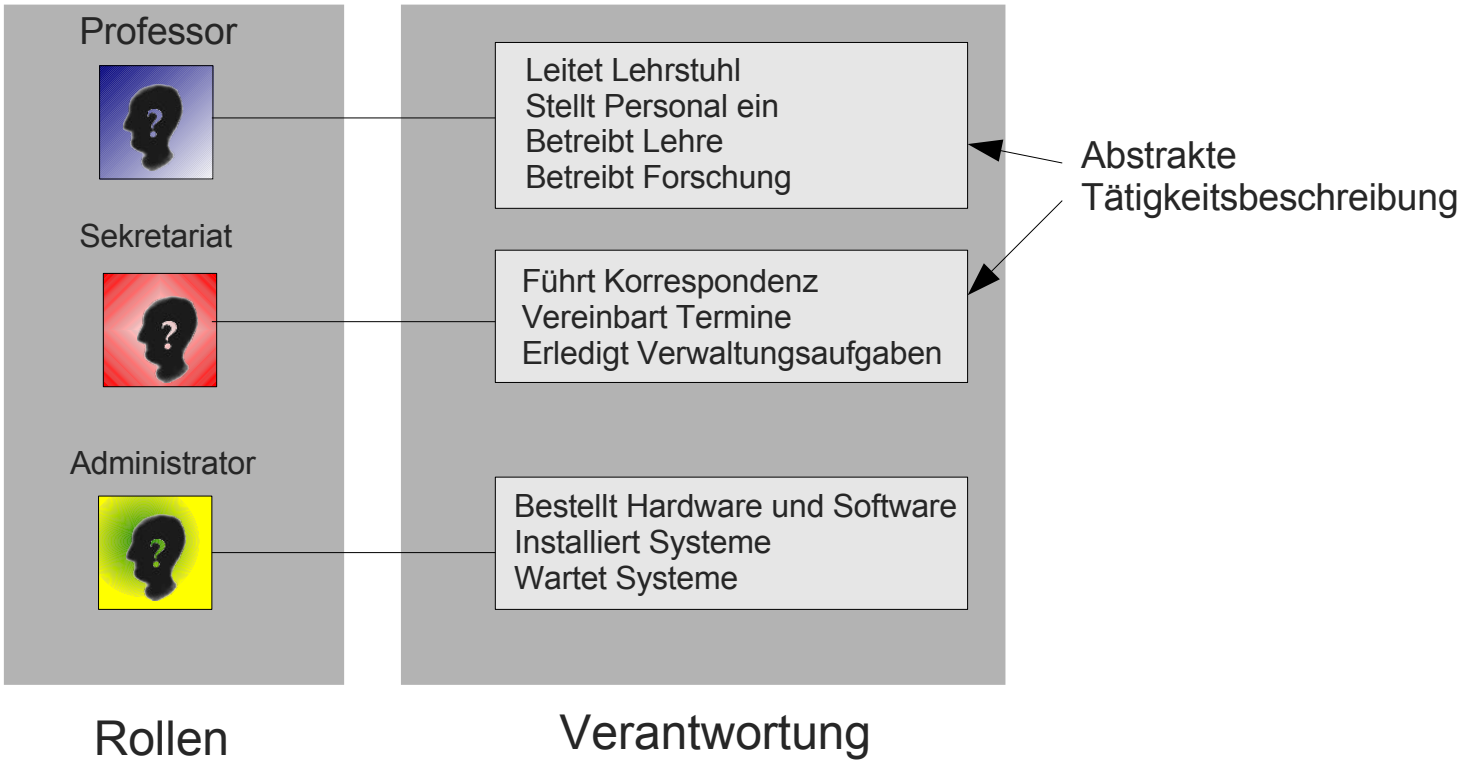
- Abstraktion in der Analysephase
  - Keine Klassen, Objekte, Agenten, Typen definieren
  - *Rolle* als abstrakter Funktionsträger
- Vergleich mit Firma
  - Rollen sind nicht ewig an konkrete Personen gebunden
  - Personen können mehrere Rollen ausfüllen
  - Konkrete Instanziierung füllt Rollen mit Personen (Leute einstellen)
  - Geeignete Vorlage für Agentengemeinschaften
    - entspechend erst später entscheiden, welchen Agenten man für diese Rolle „engagiert“

Rollen und Verantwortung

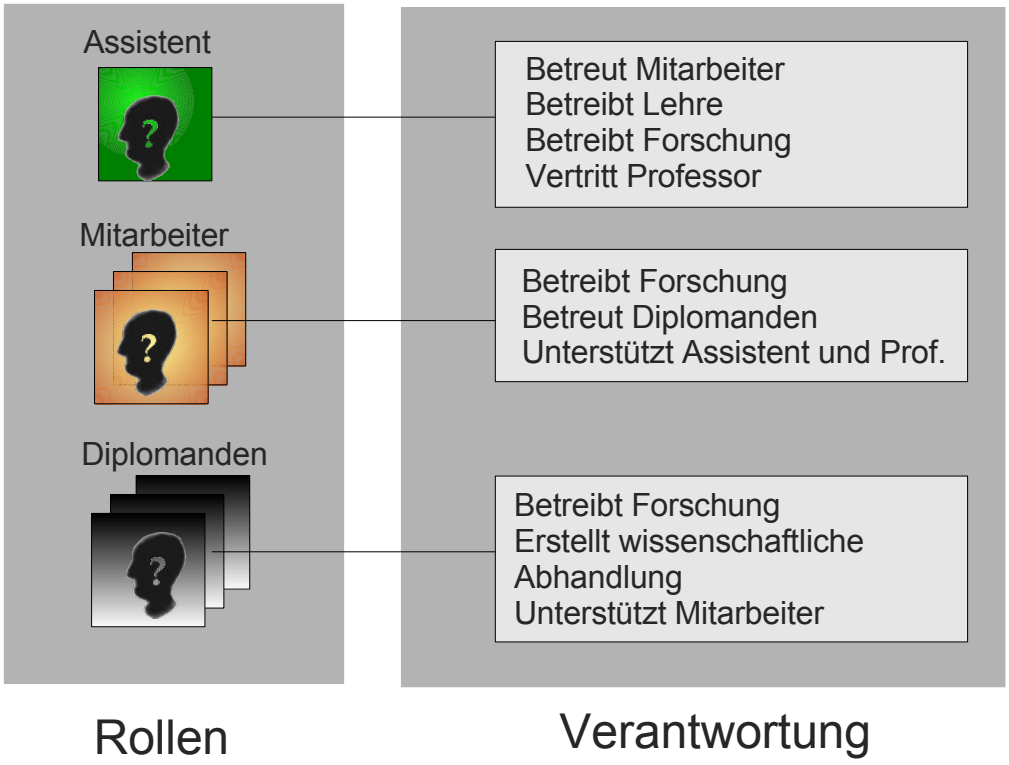
- Verantwortung als Abstraktion der Tätigkeit
  - Aus der Verantwortung erwachsen konkrete Tätigkeiten
  - ist an eine Rolle gebunden (charakterisierend)
- Beispiele
  - Administrator: muss das System in Gang halten
  - Abteilungsleiter: muss für eingehende Aufträge sorgen

Beispiel

- Im realen Leben: Besetzung eines Lehrstuhls
  - Universität erstellt Stellenausschreibung

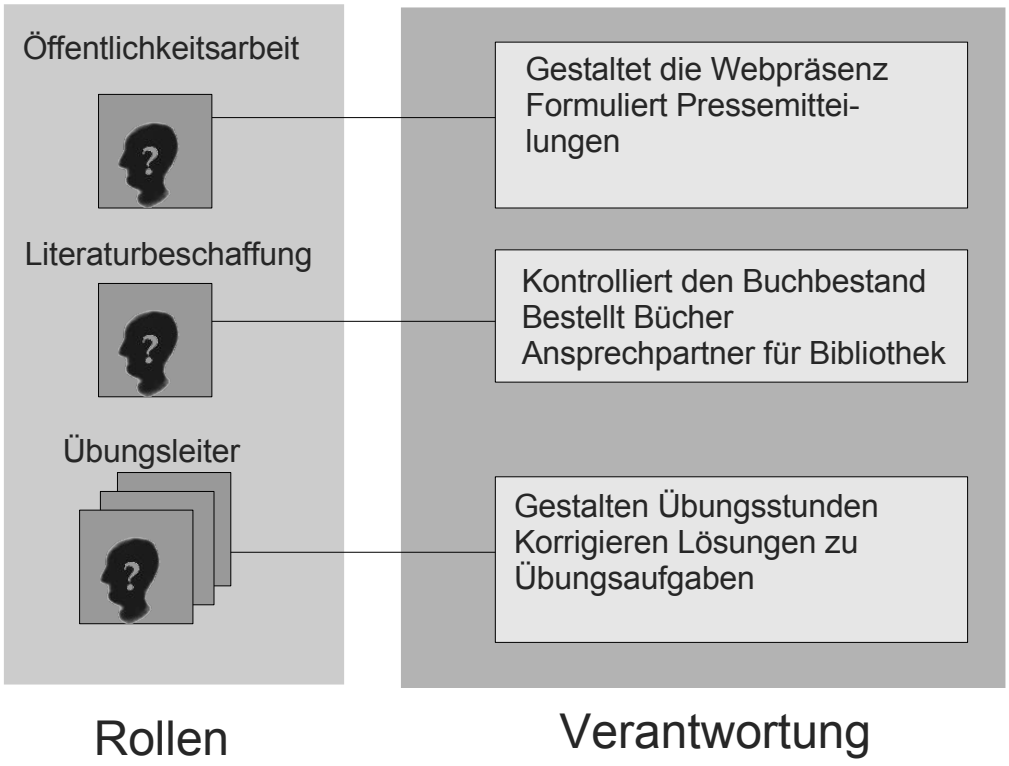


Beispiel



Rollen können quantifiziert sein (nur ein Assistent, aber mehrere Mitarbeiter)

Beispiel

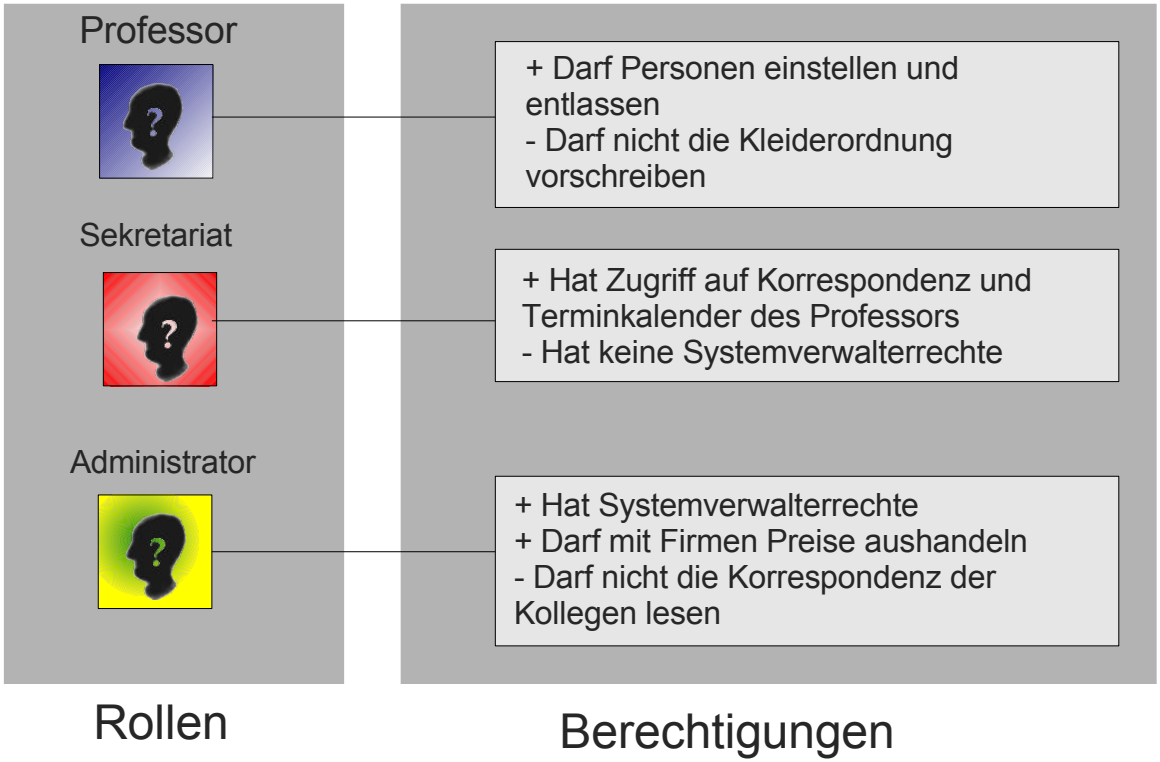


Rollen müssen nicht von speziell angestellten Personen bekleidet werden.

Berechtigungen

- Berechtigungen sind an Rollen gebunden
  - systemspezifische Rollen: Administrator (darf das System modifizieren)
  - organisatorische Rollen: Abteilungsleiter (darf Verträge abschließen)
- Zwei Aspekte der Berechtigungen
  - Welche Ressourcen dürfen zur Wahrnehmung der Rolle verwendet werden?
  - Wie viel davon darf verwendet werden?
    - Oder anders: Was darf *nicht* aufgewendet werden, um die Rolle auszufüllen?

Beispiel


Professor
Sekretariat
Administrator
+ Darf Personen einstellen und entlassen
- Darf nicht die Kleiderordnung vorschreiben
+ Hat Zugriff auf Korrespondenz und Terminkalender des Professors
- Hat keine Systemverwalterrechte
+ Hat Systemverwalterrechte
+ Darf mit Firmen Preise aushandeln
- Darf nicht die Korrespondenz der Kollegen lesen
Rollen
Berechtigungen

Berechtigungen

- Berechtigungen betreffen Nutzung von Ressourcen
  - Ressourcen sind Personen, Geld, Waren, Informationen, ...
- in Gaia
  - Ressourcen sind Informationen (Kenntnisse der Agenten)
  - Berechtigungen sind damit geeignet,
    - eine bestimmte Informationsquelle zu nutzen
    - Informationen zu ändern

Spätere Gaia-Versionen könnten ein reichhaltigeres Ressourcen-modell definieren.

Wooldridge

Beispiel „Kaffeeagent“

- Formale Darstellung von Berechtigungen nötig

Rolle: Kaffeekocher

sorgt dafür, dass die Kaffeekanne immer voll ist

Berechtigungen

reads Kaffeestatus // voll oder leer
changes Kaffeevorrat

Agent hat das Recht, den Status zu lesen und den Vorrat zu ändern

Berechtigungen (mit Parametern)

reads **supplied** Gerät
reads Kaffeestatus
changes Kaffeevorrat

Parameter „Gerät“ von anderem Agenten geliefert (also nicht festgeschrieben)

Verantwortung

- Zwei Kategorien von Verantwortung
  - Fortschritt (Liveness): etwas Sinnvolles passiert
  - Funktionssicherheit (Safety): nichts Gefährliches passiert
- Beispiel für Fortschritt beim Kaffeekocher
  - Mache Kaffee, wenn die Kanne leer ist
  - Teile den Kollegen mit, wenn der Kaffee durchgelaufen ist
- Fortschrittsangaben werden formal notiert
  - definieren damit den „Lebenszyklus“ einer Rolle
    - mögliche Ausführungspfade bei Annahme der Rolle
  - Beschreibung ähnlich den regulären Ausdrücken

    ROLLENNAME = Ausdruck

Fortschrittsspezifikation

Operatoren:	x.y	x, dann y	x^{ω}	unbegrenzt oft x
	x|y	x oder y	[x]	eventuell x
	x*	mehrmals x (auch 0,1)	x||y	x und y parallel
	x+	mind. einmal x		

Omega-Notation erlaubt die Formulierung nichtabbrechender Berechnungen.

- Kaffeekocherrolle
  - KAFFEEKOCHER = (Fülle. InformiereKollegen. PrüfeVorrat. WarteAufLeereKanne)$^{\omega}$

Protokoll
Vorgang, welche Interaktion mit anderen Agenten erfordert

Aktivität
ähnlich einer Methode/Prozedur
erfordert keine Interaktion mit anderen Agenten
(stets unterstrichen)

Sicherheitsspezifikation

- Sicherheit (Safety)
  - Agent muss in seiner Rolle gewisse Invarianten einhalten
    - z.B. niemals den Kontostand unter 100 Euro sinken lassen
  - Invariante muss so lange eingehalten werden, wie die Rolle ausgeübt wird
    - in jedem Zustand des Agenten
    - Wenn die Rolle ein Omega-Ausdruck ist, dann immer gültig
  - Beispiel Kaffeekocher
    - Kaffeevorrat > 0 (meist als Punktaufzählung dargestellt)

Weiteres Vorgehen

- Rollenmodell definieren
  - Rollenschema für jede Rolle im System definieren
    - Rollenbeschreibung
    - Protokolle und Aktivitäten
    - Berechtigungen
    - Verantwortung (Fortschritt, Sicherheit)

Vorlage für ein Rollenschema

Rollenschema	*Name der Rolle*
Beschreibung	*Umgangssprachliche Beschreibung*
Protokolle/Aktivitäten	*relevante Protokolle und Aktiv. für diese Rolle*
Berechtigungen	*dieser Rolle zugebilligte Rechte*
Verantwortung	
Fortschritt	*Fortschrittsangaben*
Sicherheit	*Funktionssicherheitsangaben*

Instanziierung des Schemas

Rollenschema: KAFFEEKOCHER

Beschreibung: Diese Rolle sichert zu, dass die Kaffeekanne nachgefüllt wird und dass die Kollegen davon erfahren.

Protokolle/Aktivitäten: Fülle, InformiereKollegen, PrüfeVorrat, WarteAufLeereKanne

Berechtigungen:

reads	**supplied** Gerät	// Name des Geräts
	Kaffeestatus	// voll oder leer
changes	Kaffeevorrat	// Kaffee noch vorhanden

Verantwortung

Fortschritt: $\text{KAFFEEKOCHER} = (\text{Fülle.InformiereKollegen. PrüfeVorrat. WarteAufLeereKanne})^{\omega}$

Sicherheit:
- Kaffeevorrat > 0

Interaktionsmodell

- Rollen sind untereinander mit Interaktionen verbunden
  - sogar ein zentrales Konzept der Rolle in Gemeinschaften
- Kommunikationsvorgänge müssen bereits in der Analysephase erfasst werden
- Interaktionsmodell: Menge von Protokolldefinitionen
- Protokolldefinitionen
  - Protokoll als eine „institutionalisierte“ Interaktion
  - Genaue Zusammensetzung und Verwendung von Nachrichten zunächst uninteressant
  - stattdessen Fokus auf Sinn und Zweck der Interaktion
  - Abhängigkeiten zwischen Interaktionen feststellbar (anhand der benötigten und gelieferten Informationen)

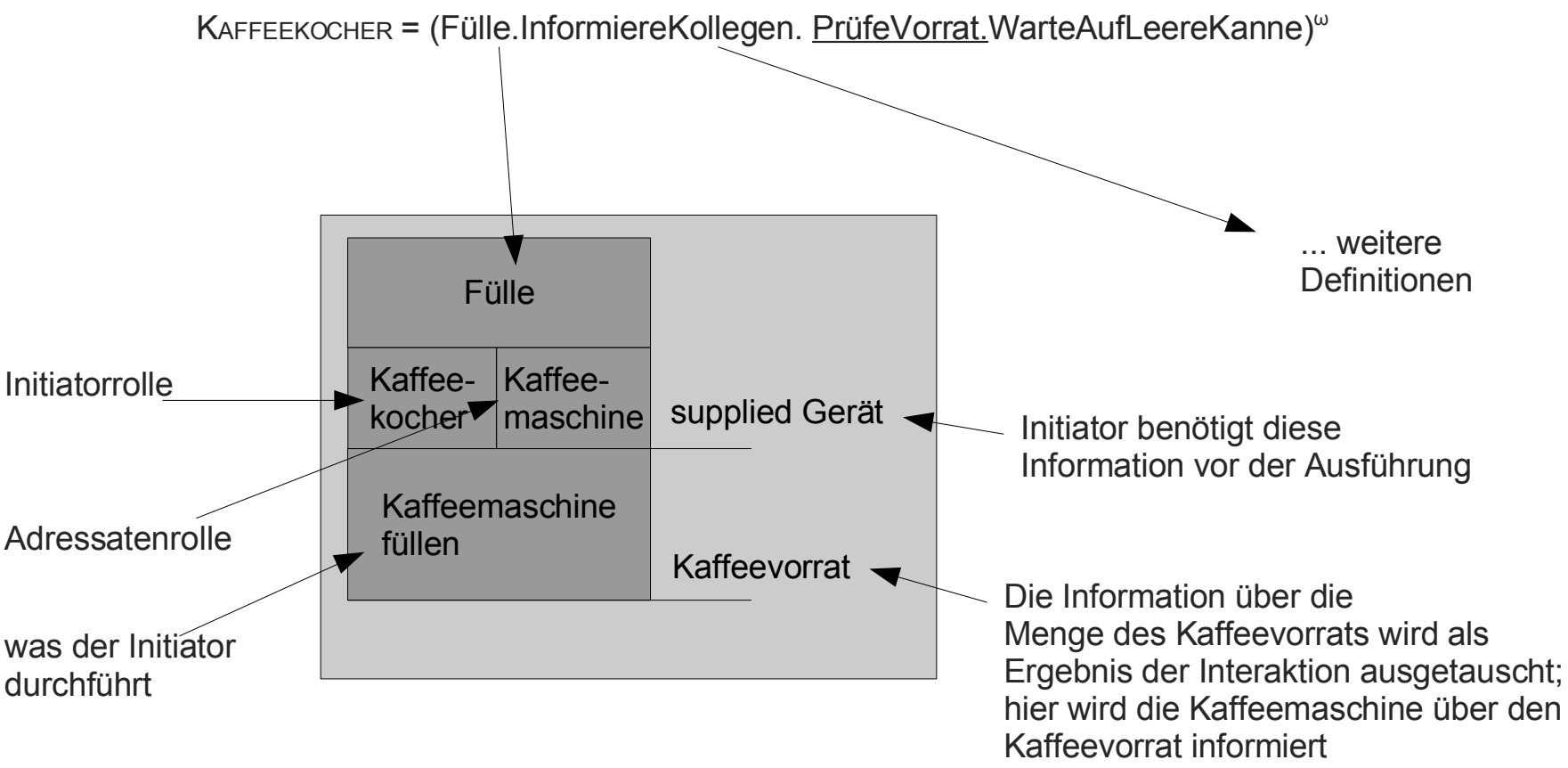
Interaktionsmodell

- Für jede Interaktion zwischen Rollen wird eine Protokolldefinition benötigt
- Eine Protokoll löst zur Laufzeit eine Menge von Kommunikationsvorgängen aus
- Beispiel Auktion
  - Protokoll = Englische Auktion
  - Erforderliche Rollen: Bieter, Verkäufer
  - Mögliche Kommunikationsmuster: Preisankündigung, Gebot
- Detailgrad sollte in der Analysephase nicht zu hoch sein

Protokolldefinition in Gaia

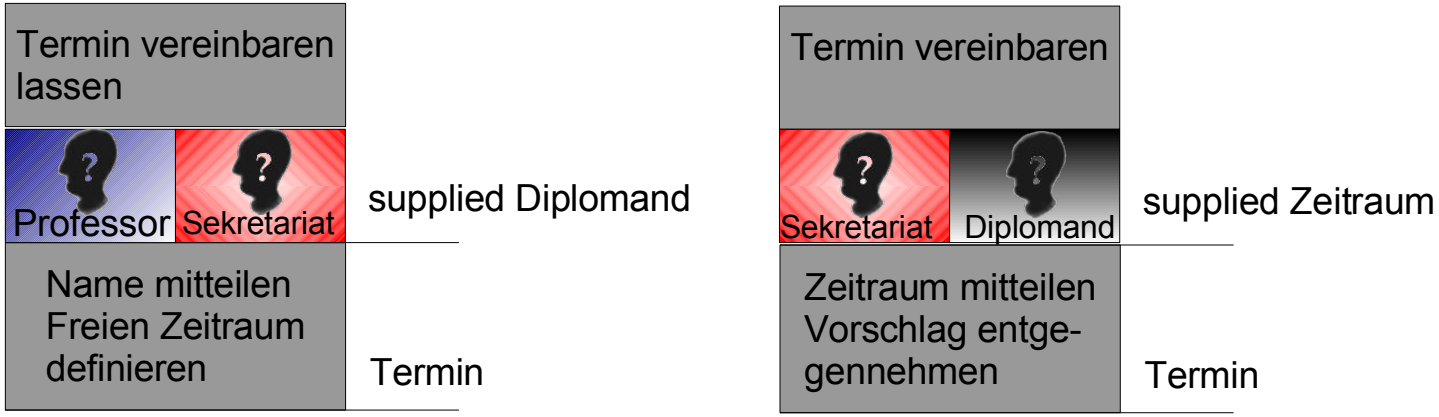
- **Zweck**: textuelle Beschreibung
- **Initiator**: Bezeichnung der Rolle(n), welche die Interaktion startet/n
- **Adressat**: Bezeichnung der Rolle(n), welche vom Initiator angesprochen werden
- **Eingaben**: Informationen, die der Initiator beim Start der Interaktion verwendet
- **Ausgaben**: Informationen, die im Laufe der Interaktion an den/die Adressaten gehen oder von dem/den Adressaten kommen
  - Das Protokoll kann als Ergebnis einen Informationsaustausch in eine oder beide Richtungen haben
- **Ausführung**: textuelle Beschreibung der vom Initiator während der Interaktion duchgeführten Aktionen

Protokolldefinition



Protokolldefinition

- Beispiel: Professur
  - Vereinbaren eines Gesprächstermins mit einem Diplomanden

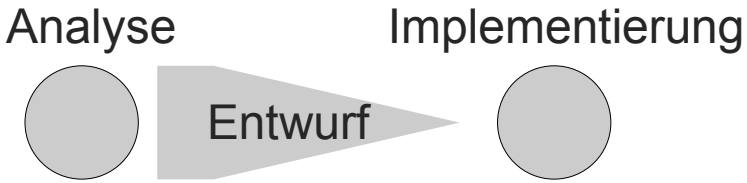


Normalerweise würde man die Tätigkeitsbeschreibung nicht so detailliert gestalten. Entsprechend sollte man beim Agentenentwurf auch nicht allzu spezifisch zu diesem Zeitpunkt sein.

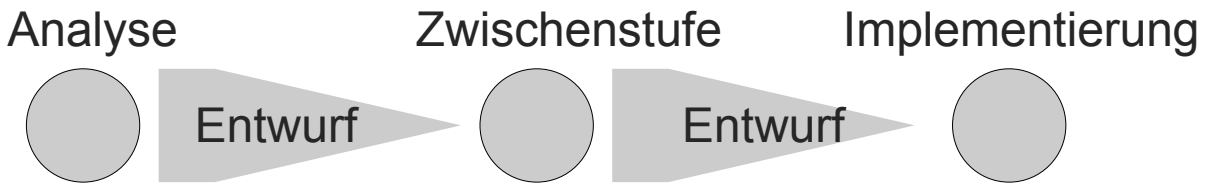
Analysephase im Überblick

- Rollen identifizieren
  - Individuen, Abteilungen, Organisationen
  - Ergebnis: vorläufiges Rollenmodell mit einfacher Beschreibung
- Protokolle identifizieren je Rolle
  - Interaktionsmuster
  - Ergebnis: Interaktionsmodell, das wiederkehrende Muster von Interaktionen zwischen Rollen erfasst
- Mithilfe der Protokolle die Rollen ausarbeiten
  - Ergebnis: Ausführliches Rollenmodell mit Berechtigungen, Funktionen, Protokollen, Aktivitäten
- Gegebenenfalls wiederholen

Entwurfsphase



- Traditionell
  - fortschreitende Konkretisierung
  - erlaubt am Ende eine leichte Implementierung
- Anders bei Gaia
  - ebenfalls Konkretisierung
  - danach jedoch Anwendung traditioneller Entwurfstechniken zur Erstellung der Anwendung (z.B. Objektorientierung)



Entwurfsphase

- Drei Modelle beherrschen die Entwurfsphase
  - Agentenmodell
    - besteht aus Agententypen, die das System bestimmen, und den Agenteninstanzen, die von diesen Typen erzeugt werden
  - Dienstmodell*
    - Funktionalitäten, die realisiert werden müssen, um die Agentenrolle zu realisieren
  - Bekanntschaftsmodell
    - dokumentiert den Kommunikationsfluss zwischen den Agenten

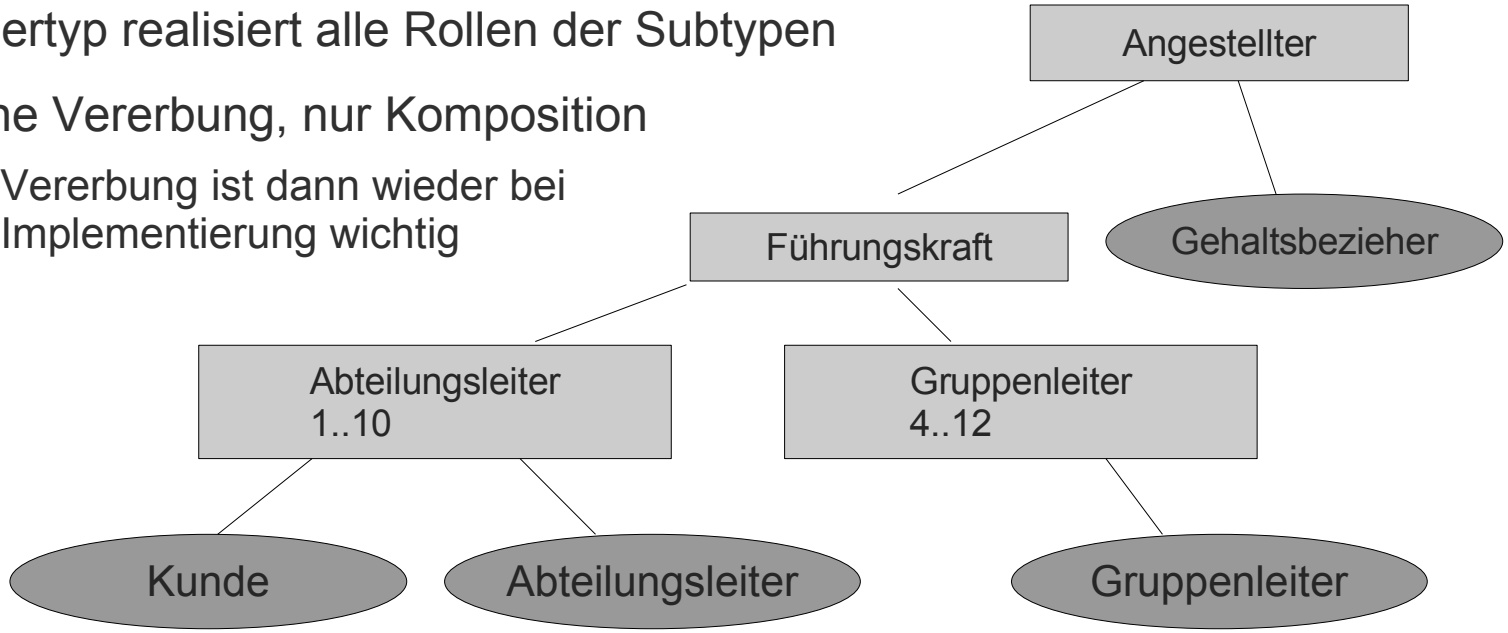
*nicht „Dienste“ im Gegensatz zu Agenten

Agentenmodell

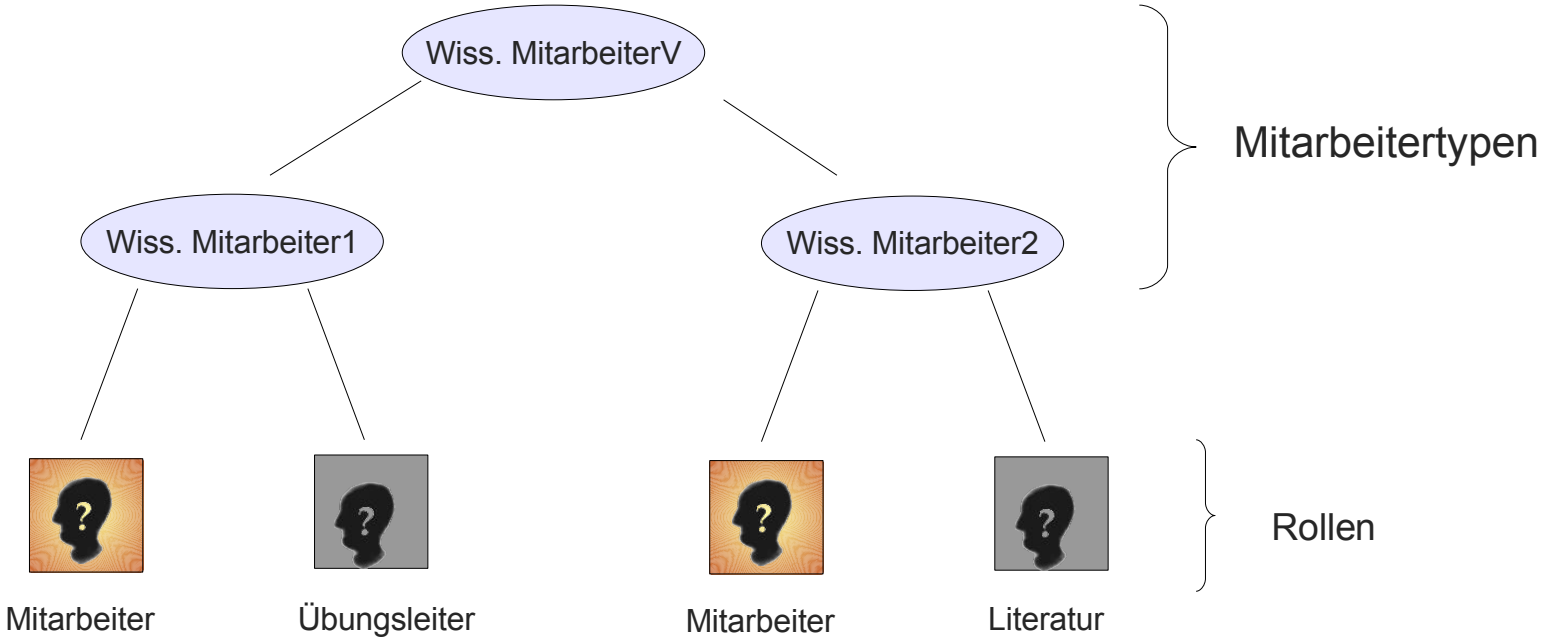
- Agententyp
  - Menge von Agentenrollen
    - Ein Agententyp kann mehrere Rollen beinhalten!
    - Mögliche Ersparnis eines eigenständigen Agenten
    - Kohärenz des Typs nicht gefährden, d.h. die Semantik nicht „verwässern“ (die eierlegende Wollmilchsau passt überall)
  - Quantifizierung der Typinstanzen: m, m..n, *, +

Agentenmodell

- Repräsentiert durch einen Typbaum
  - Blätter sind Rollen
  - Innere Knoten sind Agententypen
  - Supertyp realisiert alle Rollen der Subtypen
  - Keine Vererbung, nur Komposition
    - Vererbung ist dann wieder bei Implementierung wichtig



Die Professur



Eine Instanz des Typs „Wiss. MitarbeiterV“ müsste Mitarbeiter sein, sich um die Übungen sowie um die Literaturbeschaffung kümmern.

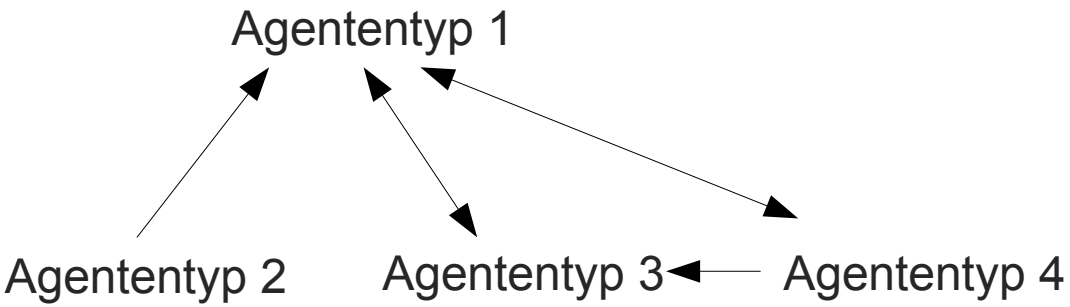
Dienstmodell

- eigentlich Funktionsmodell
  - Funktionalitäten, welche die Rolle ausmachen
  - realisiert durch agenteninterne Strukturen
  - Jede Aktivität aus der Analysephase wird durch eine Funktionalität realisiert, aber nicht umgekehrt
  - abzuleiten aus Aktivitäten, Protokollen, Fortschrittsangaben, Verantwortung
  - keine unmittelbare Vorgabe an die Implementierung

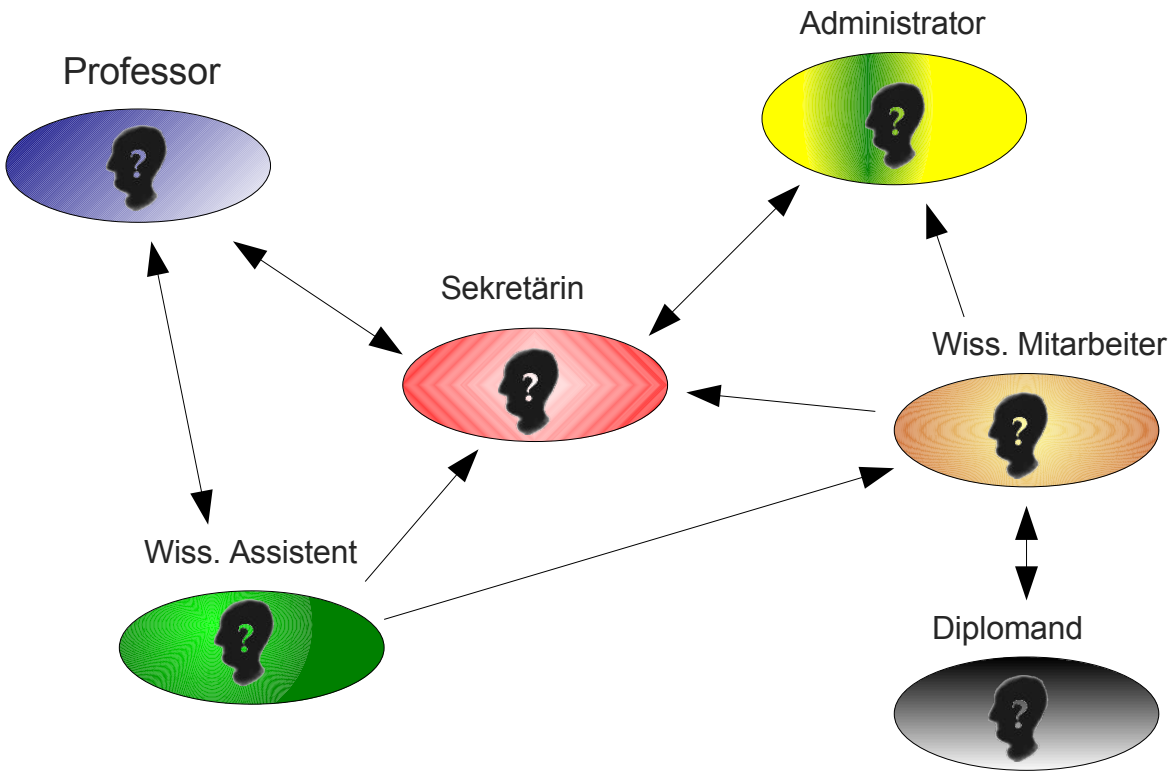
Dienst	Eingabe	Ausgabe	Vorbedingung	Nachbedingung
PrüfeVorrat	Kaffeevorrat Mindestmenge	ok/nicht ok	Kaffeevorrat>0	Kaffeevorrat>0

Bekanntschaftsmodell

- Acquaintance model
  - Soll nur die Kommunikationswege darstellen
  - Dient dem Nachweis von Engpässen
  - Kann dazu dienen, die Analyse zu verfeinern
    - z.B. Anwendung umstrukturieren, um Abhängigkeiten zu lockern
    - neue Rolle einführen ...



Bekanntschaftsmodell



Entwurf ist verbesserungswürdig:

Der Typ „Sekretärin“ ist sehr zentral; fällt sie aus, dann können die Mitarbeiter sich nicht mehr an ihren Chef wenden. Außerdem können die Mitarbeiter nur über den Umweg Sekretärin/Professor mit dem Assistenten reden.

(Unrealistisch im akademischen Umfeld, aber in manchen Firmen oder Behörden gibt es durchaus die „langen Wege“)

Entwurfsprozess: Zusammenfassung

- Agentenmodell generieren
  - Rollen identifizieren, Rollen in Typen sammeln; Typgraph bauen
  - Instanzen der Agententypen mittels Annotationen dokumentieren (eine, viele usw.)
- Dienstmodell generieren
  - Aktivitäten, Protokolle, Sicherheit, Fortschritt je Rolle analysieren
- Bekanntschaftsmodell generieren
  - gewonnen aus dem Interaktions- und Agentenmodell

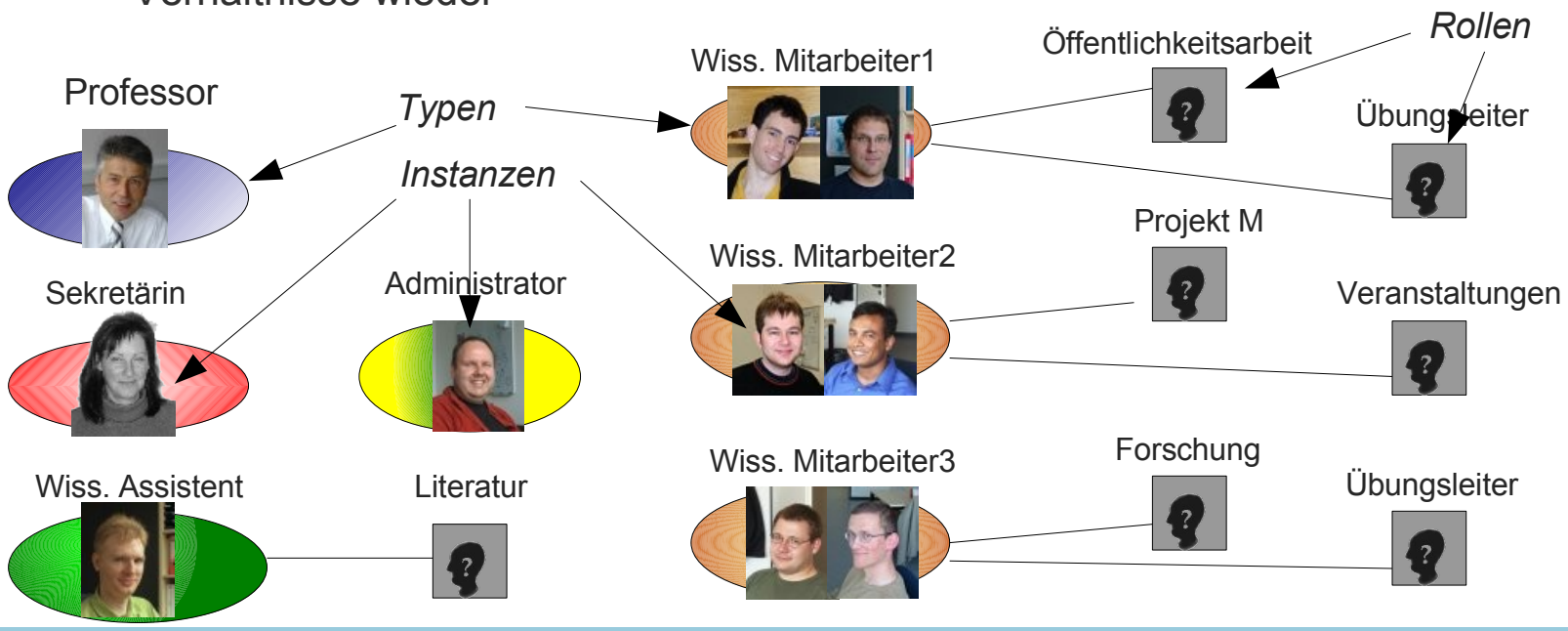
Implementierung

- Phase der Implementierung außerhalb von Gaia
  - Kann mit traditionellen Mitteln durchgeführt werden
- Im Beispiel der Professur: Reale Leute einstellen
- Entsprechend bei Agentensystemen
  - Agenten werden als „Besetzung“ von „Posten“ verstanden, nicht mehr als einfache Objekte, die einen Dienst anbieten
  - Eignung von Agenten durch sein Rollenangebot definiert

In der realen Welt hingegen geht man nicht von verschiedenen Menschentypen aus, denen bestimmte Rollen fest vorgegeben sind

Beispiel

- Besetzung einer Professur
  - Rolleneinteilung teilweise fest, teilweise flexibel
    - Fähigkeit, eine Rolle anzunehmen, wird aber vorausgesetzt
    - Das Beispiel führt nicht alle Rollen auf und spiegelt nicht die tatsächlichen Verhältnisse wieder



Gaia

- Anschauliches Beispiel in

  Wooldridge, Jennings, Kinny: *The Gaia Methodology for Agent-Oriented Analysis and Design*
  Proceedings of AAMAS (Autonomous Agents and Multi-Agent Systems) 2000
  Kluwer Academic publishers, pp. 285-312

  http://www.ecs.soton.ac.uk/~nrj/download-files/jaamas2000.pdf

AgentUML

- Zusammenarbeit der OMG und FIPA
- UML
  - bewährt zur Modellierung von Anwendungen
  - Mängel beim Entwurf von Agentenanwendungen
- Vorschläge aus Gaia übernommen
  - Protokolle als Baustein repräsentieren

Achtung:
UML-Grundkenntnisse
im Folgenden vorteilhaft

Unified Modeling Language

- Elemente
  - **Use case (Anwendungsfall)**: Spezifikation der Aktionen des Systems mit externen Aktoren. Beschreibt, wie ein Anwender mit einem Softwareprodukt umgehen kann.
  - **Statische Modelle**: Beschreiben die festgelegte Semantik von Daten und Nachrichten in einer konzeptionellen und operationellen Weise (u.a. Klassendiagramme)
  - **Dynamische Modelle**: Beinhalten Interaktionsdiagramme (d.h. Sequenz- und Kollaborationsdiagramme), Zustandsautomaten, Aktivitätsdiagramme
  - **Installationsmodelle**: Beschreiben die Verteilung von Komponenten auf verschiedenen Plattformen (deployment)
  - **Object Constraint Language (OCL)**: Einfache formale Sprache, um weiter gehende Semantik in UML zu formulieren (Invariante, Vor-, Nachbedingungen von Operationen in einem Objektnetz)

Konzeptionelle Schwächen

- Schwächen des bisherigen Ansatzes
  - Agenten sind aktiv und übernehmen die Initiative; dies ist nicht adäquat repräsentiert
  - Agenten können Anfragen ablehnen, abhängig von ihren eigenen Plänen
  - Agenten handeln zumeist in Kooperation mit anderen Agenten

Spezifische Schwächen

- Interaktionsdiagramme
  - richten sich eher auf Methodenaufrufe zwischen Objekte
  - können komplexe soziale Interaktionen nicht übersichtlich genug darstellen
- Zustandsdiagramme
  - eher geeignet für einzelne Objekte in unterschiedlichen Szenarien; Beschreibung einer Gruppe schwierig
- Aktivitäts- und Klassendiagramme
  - können mit geringfügigen Erweiterungen verwendet werden

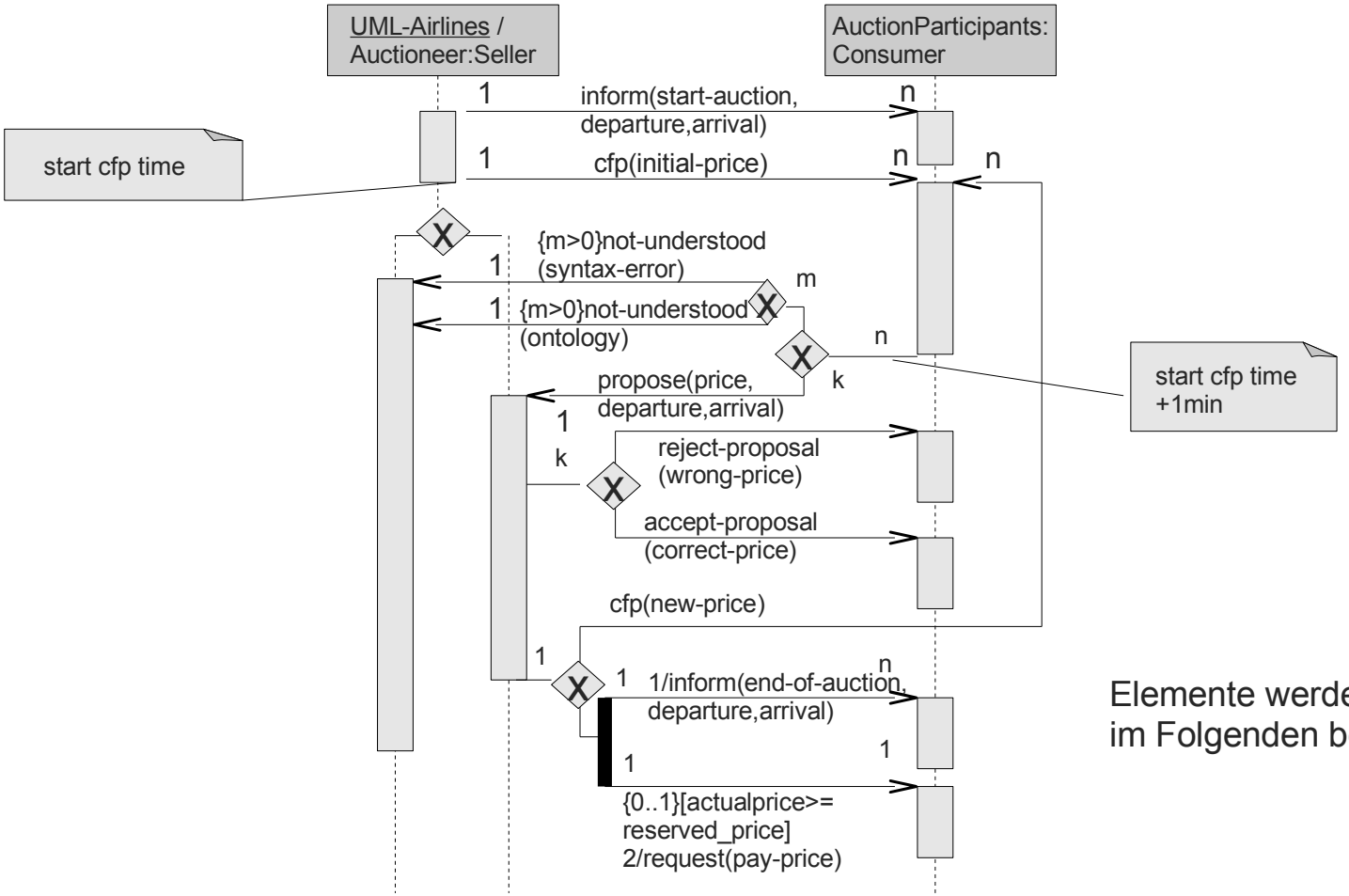
Ansatz der AgentUML

- Drei Stufen
  - Stufe 1: Gesamtprotokoll
    - Kommunikationsmuster
    - Pakete und Schablonen der UML nutzen
  - Stufe 2: Interaktionen zwischen Agenten
    - Sequenzdiagramme (erweitert für Protokolle)
    - Kollaborationsdiagramme
    - Aktivitätsdiagramme
    - Zustandsautomaten
  - Stufe 3: Vorgänge innerhalb der Agenten
    - Aktivitätsdiagramme
    - Zustandsdiagramme

Gesamtprotokoll

- Beschreibung
  - der zulässigen Nachrichtenfolgen
  - der Vorbedingungen und inhaltlichen Nebenbedingungen
  - der Semantik gemäß den Kommunikationsakten innerhalb des Kommunikationsmusters
- Neuer Diagrammtyp: *Protokolldiagramme*
  - Kombination aus Sequenzdiagramm und Zustandsdiagramm
- Elemente der Protokolldiagramme
  - Agentenrolle
  - Lebenslinie
  - Interaktionsvorgänge
  - Protokollbausteine
  - Nachrichten
  - Protokollschablonen

Beispiel eines Protokolldiagramms



Elemente werden
im Folgenden beschrieben

Agentenrolle

- Rolle *agent-role*
  - Menge von Agenten mit gleichen Eigenschaften (auch als *Typ* zu bezeichnen), z.B. Schnittstellen, Dienste
  - Agenten können mehrere Rollen realisieren (auch innerhalb einer Interaktion)
  - Protokolldefinition kann sich auf einzelne Instanzen oder auf die Rolle an sich beziehen

Agentenrolle

Instanz1 Instanz2 ... InstanzN / Rolle1 Rolle2 ... RolleM: Klasse

Instanzen, Rollen und Klassen können weggelassen werden

Die Unterstreichung dient der optischen Trennung von Instanzen und Rollen. Sind keine Instanzen angegeben, wird nichts unterstrichen

Beispiele:

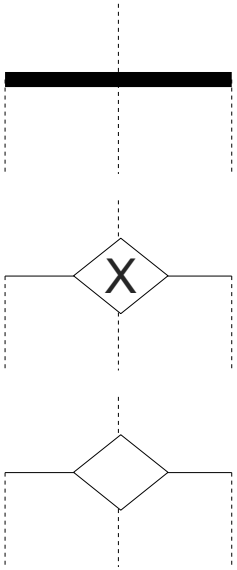
a) MichaelsAgent / Kaufagent Verkaufsagent : BuyAndSellAgent

b) Vermittler : TraderAgent (Kurzform ohne Instanzen)

Lebenslinien

- Lebenslinie
  - jeweils einem Agenten zugeordnet, als gepunktete Linie
  - beschreibt die Zeit, während welcher der Agent „lebt“, also zwischen Erzeugung und Entsorgung
  - kann in mehrere Äste aufgespalten werden
    - Parallelismus: Alle Äste (bzw. eine Auswahl) werden ausgeführt
    - Entscheidung: Genau ein Ast wird ausgeführt
  - können an spezifischen Punkten wieder zusammengeführt werden
  - Aufspaltung korrespondiert mit entsprechendem Agentenverhalten und Reaktion auf eintreffende Nachrichten

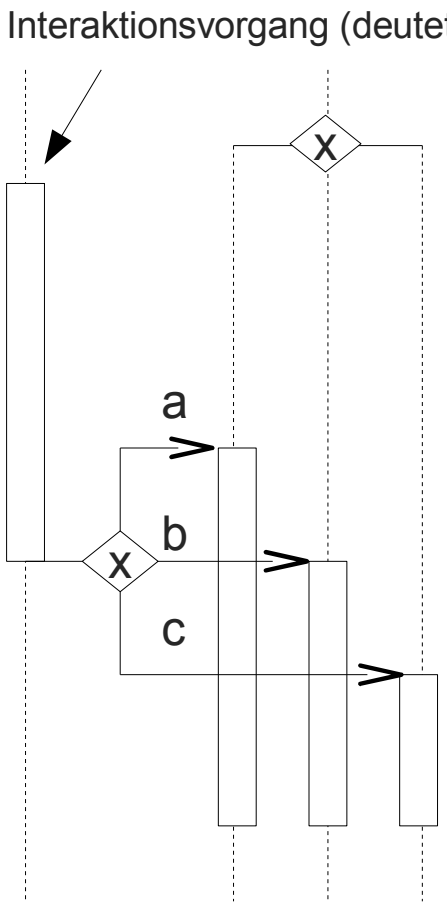
Lebenslinien



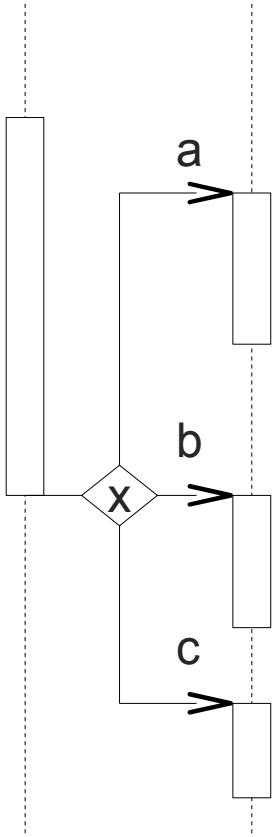
AND

XOR

OR



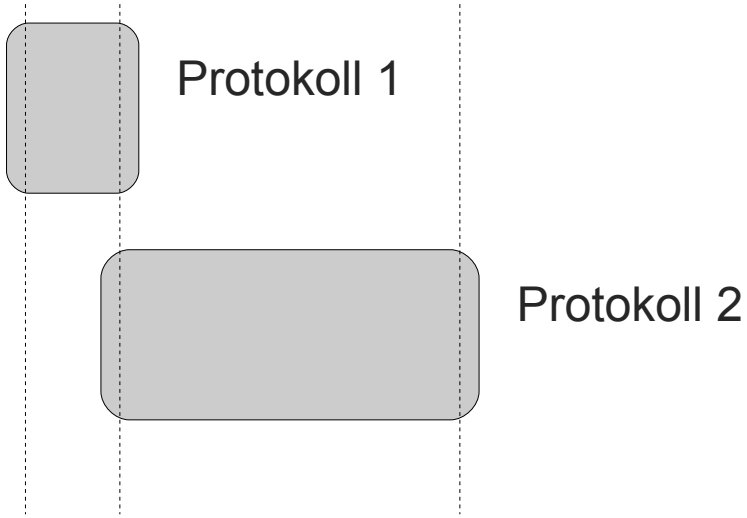
Ausführliche Form



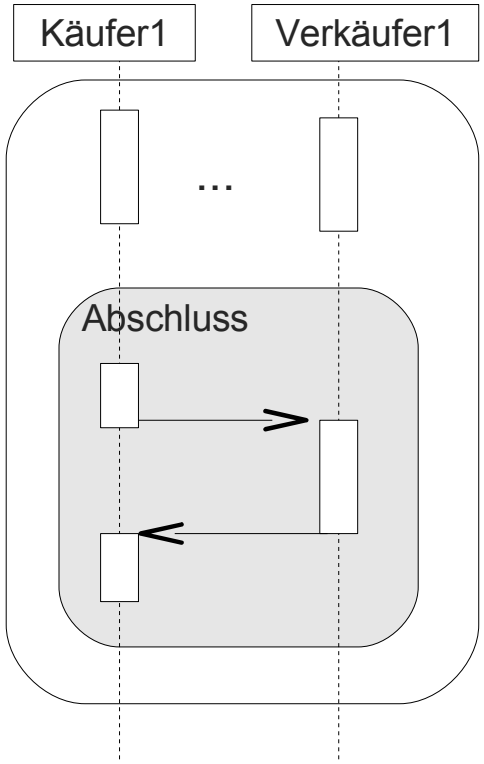
Kurzform

Protokolle als Bausteine

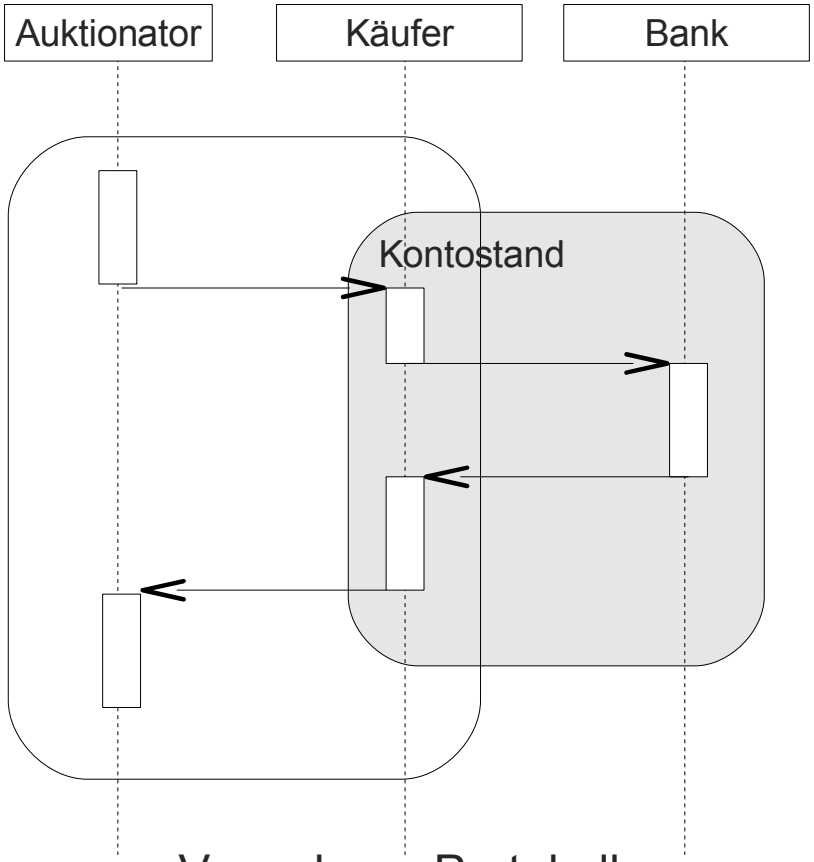
- Definition eines Protokolls
  - Danach nutzt man das Protokoll im Protokolldiagramm als fertige Box



Protokolle als Bausteine



Verschachteltes Protokoll
(kann auch wiederholt ausgeführt werden, Kennzeichnung: m..n, *)



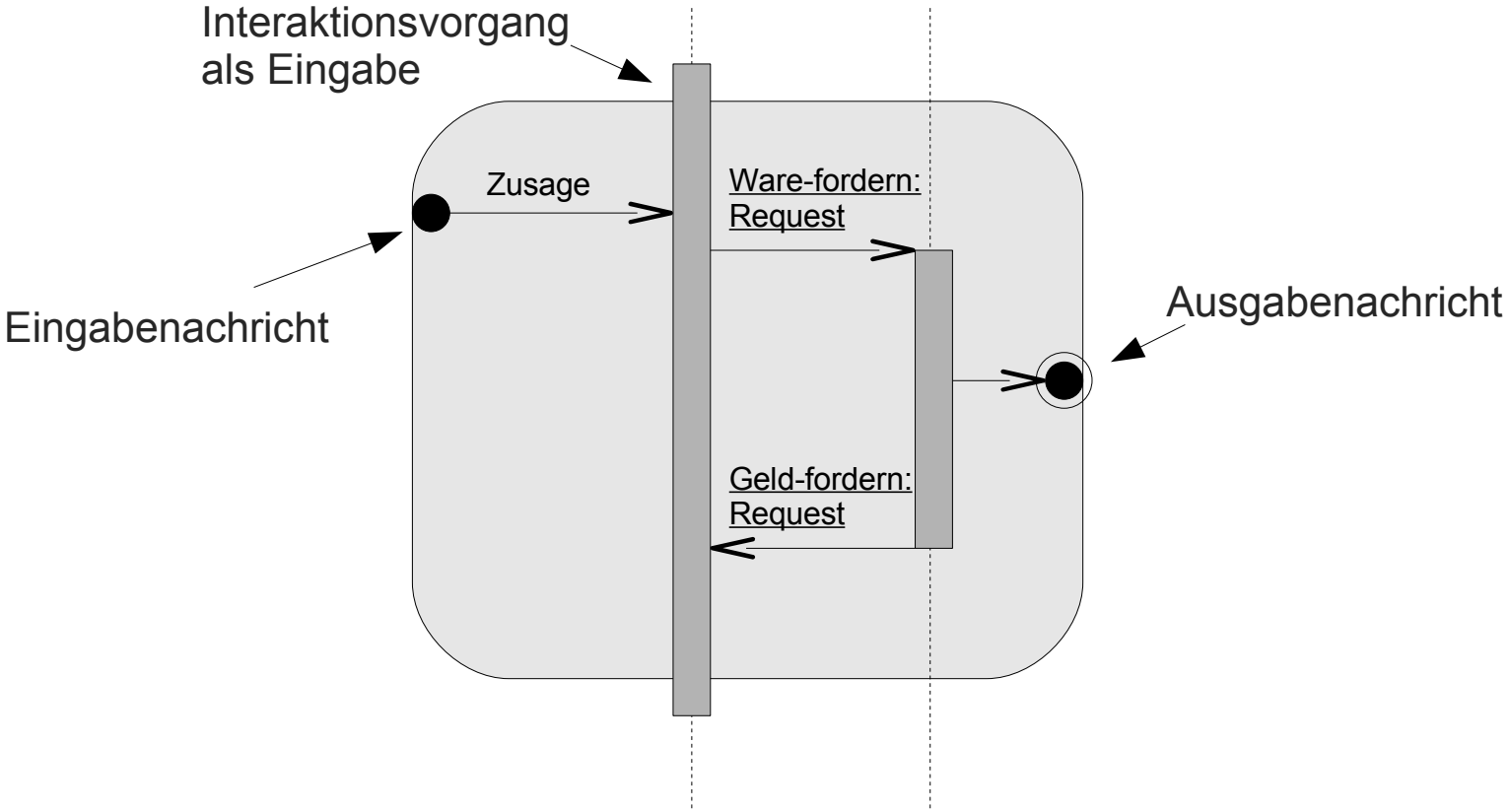
Verwobene Protokolle

Eingaben und Ausgaben

- Anwendung bei verschachtelten Protokollen
- Eingabeparameter sind
  - Interaktionsvorgänge, die im verschachtelten Protokoll weiterlaufen
  - Nachrichten, die von anderen Protokollen empfangen werden
- Ausgabeparameter sind
  - Interaktionsvorgänge, die im verschachtelten Protokoll gestartet werden und danach weiterlaufen
  - Nachrichten, die an externe Protokolle verschickt werden

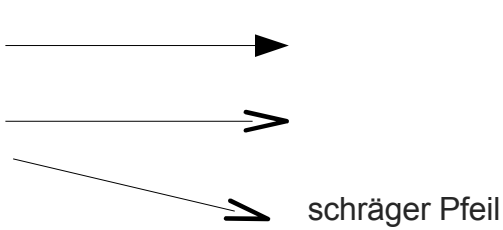
Eingaben und Ausgaben

Beispiel



Nachrichten

- Protokoll gibt zunächst nicht an, wie sich die Teilnehmer beim Austausch verhalten
- Deklaration von Synchronität und Asynchronität
  - ab UML 1.3
    - synchron:
    - asynchron:
    - mit Laufzeit:



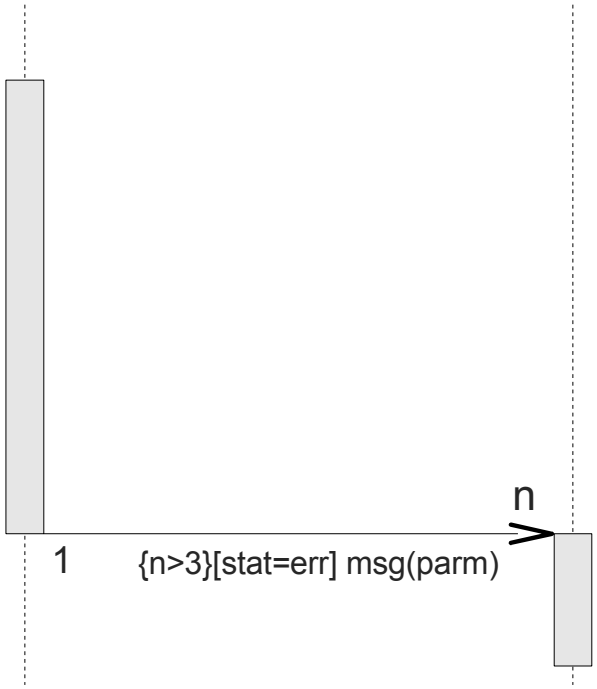
  - Bei synchronen Nachrichten wartet der Absender auf die Antwort, bevor er weiterarbeiten kann

Bei AUML zählen *kommunikative Akte* als Nachrichtenaustausch, nicht die einzelnen Nachrichten wie in der Standard-UML

Nachrichten

- Wiederholung
  - Verwendung eines Pfeils mit entsprechenden Bedingungen (guards) und Einschränkungen (constraints)
  - zeigt auf Interaktionsvorgang innerhalb des Protokolls
- Nachrichtenbezeichner, bestehend aus
  - Bezeichnung des kommunikativen Akts (etwa „cfp(new-price)“), ggf. mit Liste von Argumenten
  - Kardinalität, also die Anzahl der Empfänger
  - Einschränkungen (des Typs, z.B. { m>0 }) und Bedingungen (damit die Nachricht gesendet wird, z.B. [actualprice >= reservedprice])
  - Wiederholung m..n, *

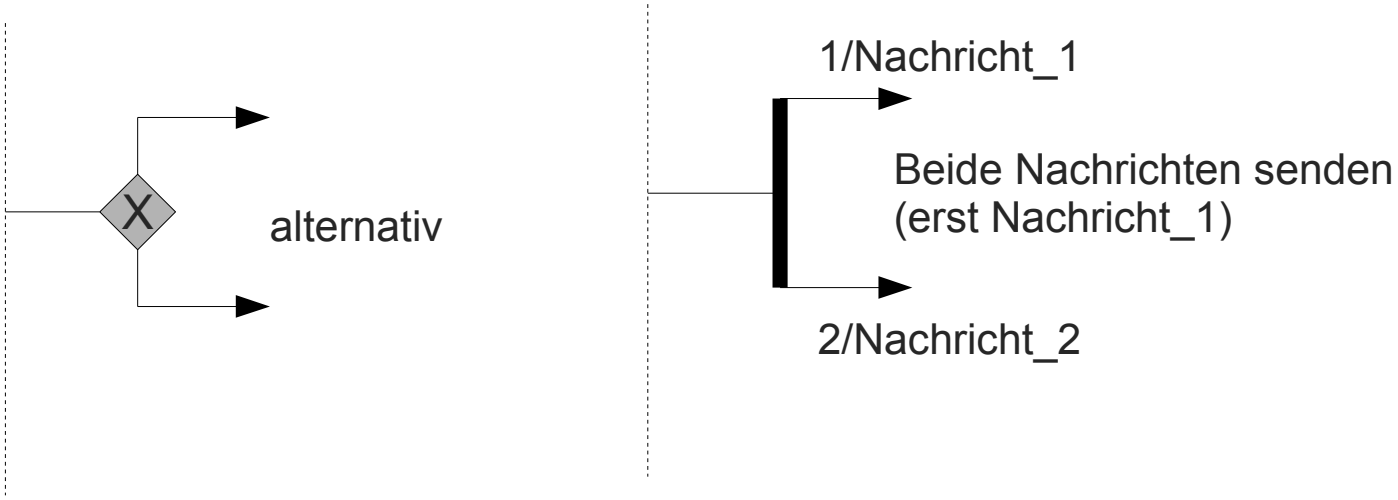
Nachrichten



Nachricht msg (mit Parameter parm) wird asynchron an n Empfänger gesendet (n ist größer als 3), wenn die Bedingung stat=err wahr ist.

Nachrichten

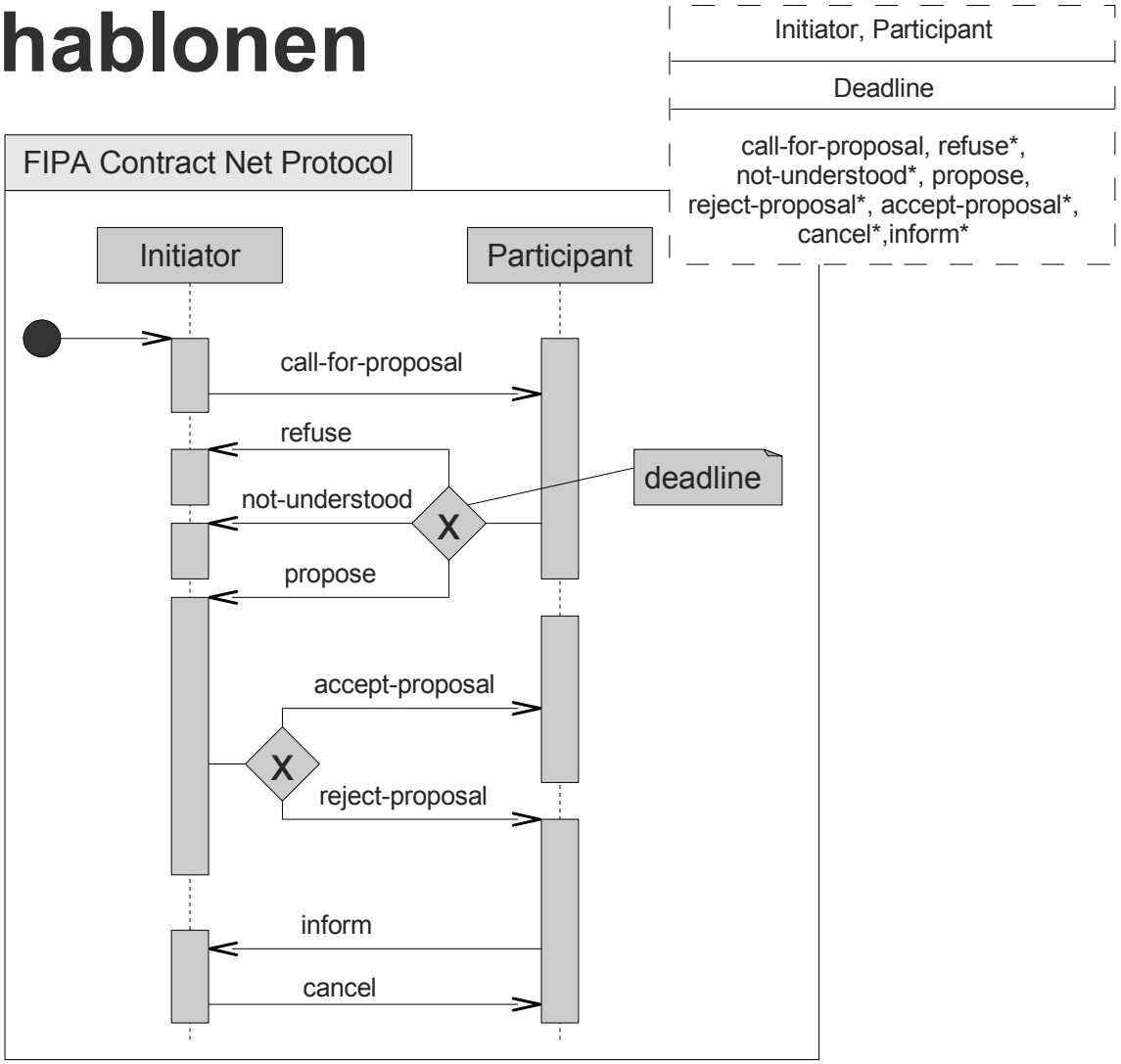
- Nachrichten können parallel oder alternativ gesendet werden
  - für parallele Sendung kann auch Reihenfolge angegeben werden



Protokollschablone

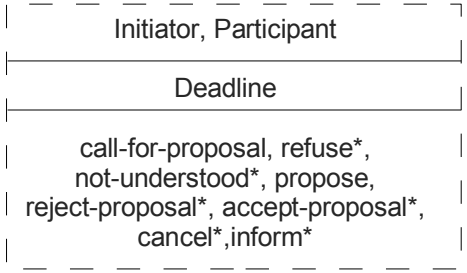
- Analogie zu Klassen
  - Protokollschablone beschreibt den prinzipiellen Vorgang
  - wird dann zu einem konkreten Protokoll instanziiert
  - Deklaration von Parametern, die durch Wertzuweisung gebunden werden
- Beispiel
  - Anfrage-Antwort-Protokollschablone
  - Auktionsprotokollschablone

Protokollschablonen

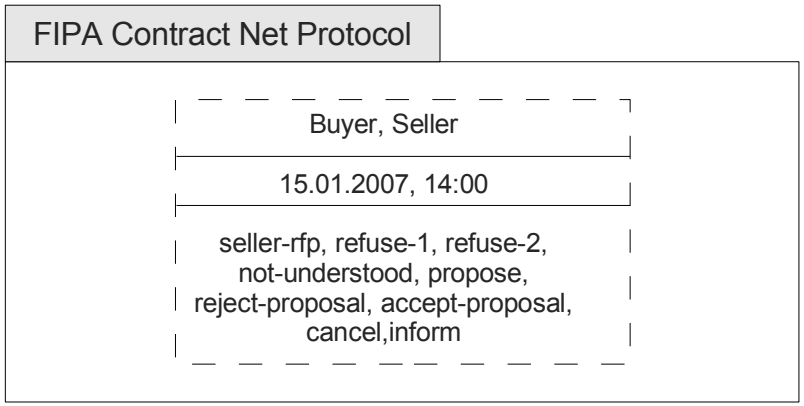


Protokollschablonen

- Schablone
  - Ein konkretes Protokoll kann hiermit „instanziiert“ werden, das die englische Auktion verwendet
  - Sternchen deuten an, dass es sich um kommunikative Akte mit verschiedenen möglichen Nachrichten handelt (verschiedenen Instanziierungen im konkreten Protokoll)

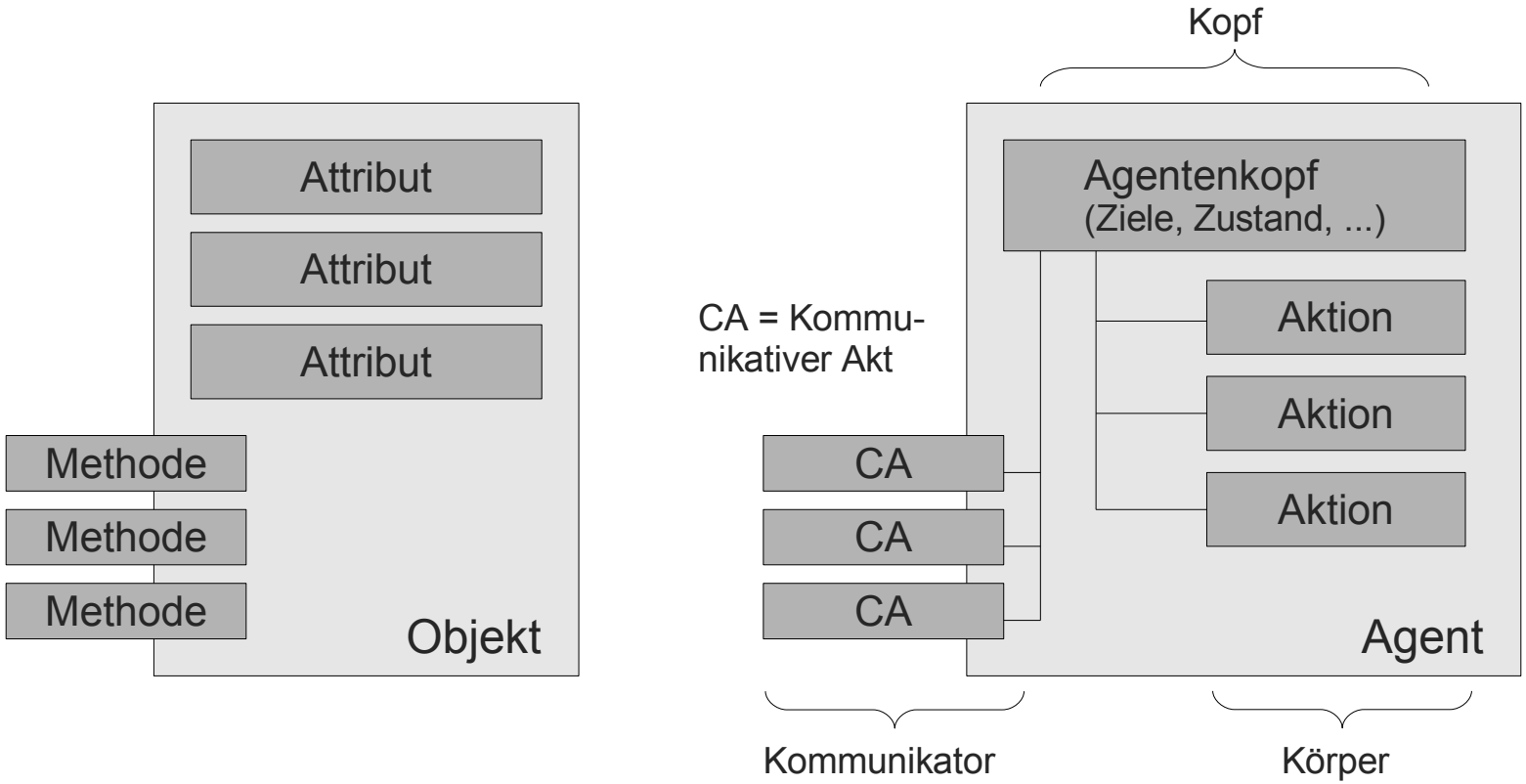


Instanziierung



Nicht UML-1.x-konform, aber mögliche Erweiterung

Vom Objekt zum Agent



Ähnlichkeiten vorhanden, aber die Unterschiede müssen geeignet in einem Klassendiagramm wiedergegeben werden können.

Vom Objekt zum Agent

- Spezifikation des Agentenverhaltens
  - muss verschiedene Strategien repräsentieren können, z.B. BDI
  - Semantik der kommunikativen Akte
  - Reaktion auf eintreffende Nachrichten
  - prozedurale oder deklarative Beschreibung
- Proaktives Verhalten
  - nicht aufgrund eintreffender Nachrichten
  - entsprechende Zustandsautomaten im Agentenkopf oder spezielle Aktionen

Vom Objekt zum Agent

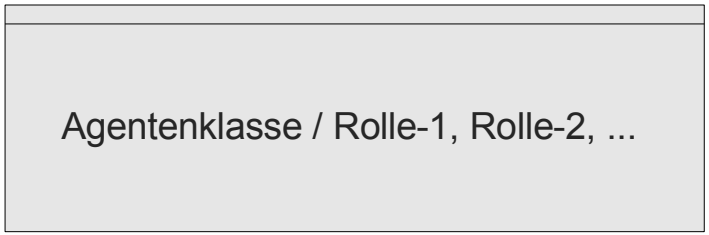
- Abstrakte Aktionen
  - definiert über Vorbedingungen, Effekte, Invarianten
- Vererbung soll möglich sein
  - von Zuständen, Aktionen, Methoden, Nachrichtenverarbeitung
- Assoziationen
  - zeigen an, dass Agent A die Dienste eines Agenten B nutzt
  - Kardinalitäten und Rollen angeben

Klasse eines Agenten

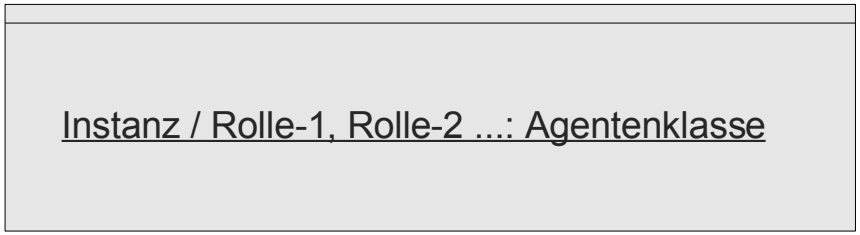
- Agentenklasse
  - entsprechend zu deuten: als Vorlage zur Erzeugung eines Agenten (möglicherweise aus vielen Objektklassen bestehend)
- Spezifikationsebenen in UML
  - **Konzeptionelle Ebene**: Agentenklasse korrespondiert mit einer bestimmten Rolle (Einkäufer, Routenplaner, ...)
  - **Spezifikations-/Schnittstellenebene**: Agentenklasse liefert Vorlage für die Agenteninstanzen (ohne Implementationsdetails)
  - **Implementierungsebene**: Agentenklasse definiert die Zusammenarbeit von Agenten und insbesondere die Implementierung (auch vom Agentenkopf)

Arten von Klassendiagrammen

Agentenklasse schlechthin



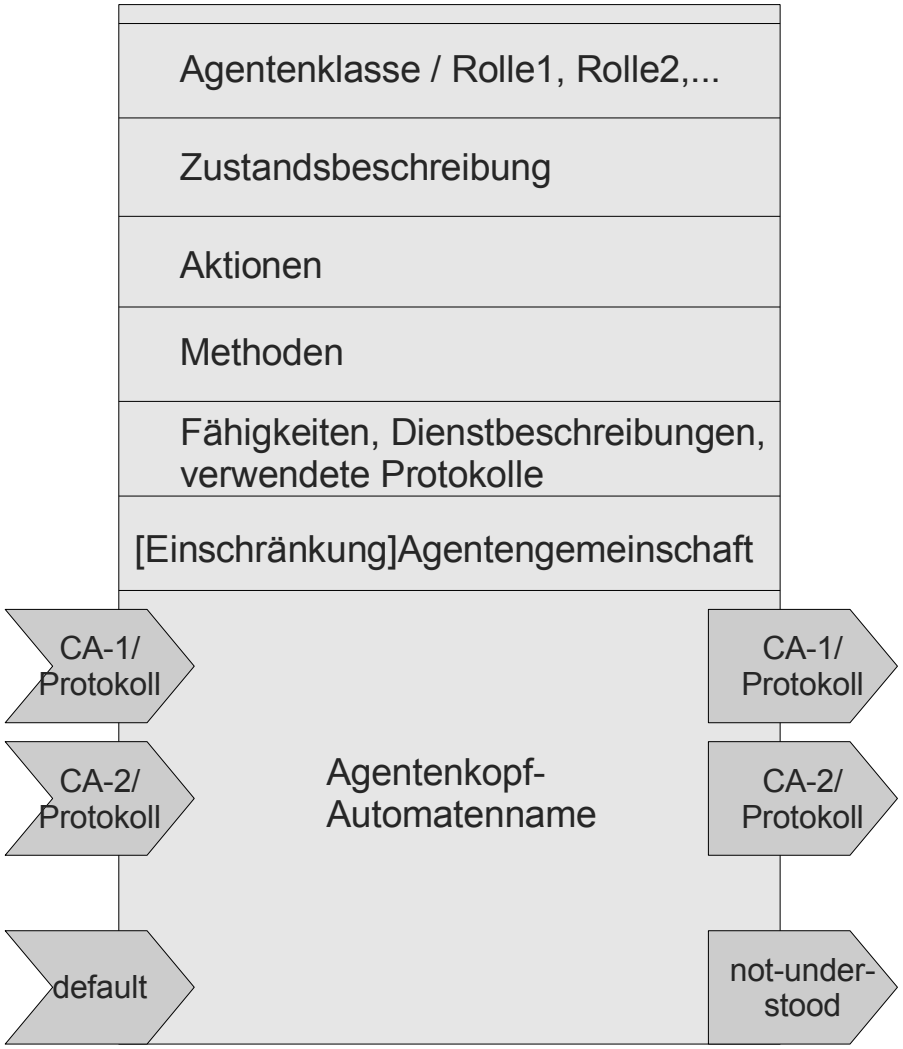
Agentenklasse, die bestimmte Rollen ausfüllt



Agenteninstanz, die bestimmte Rollen ausfüllt

Klassendiagramme

- Erweiterung der UML-Diagramme
- Kopf beinhaltet Verhalten des Agenten (in Form von Automaten)
- Standard-UML auch verwendbar; neue Diagrammform jedoch übersichtlicher
- Agentengemeinschaft: gibt an, ob und in welcher Gemeinschaft dieser Agent betrieben wird



Zustandsbeschreibung

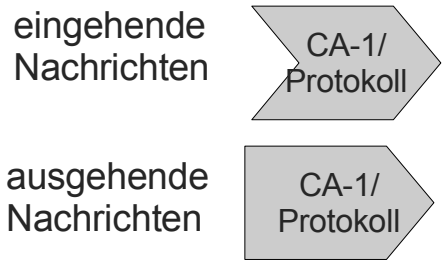
- ähnlich wie Feldbeschreibungen in Klassendiagrammen
  - jedoch mit neuer Klasse „wff“: well-formed formula (für jegliche Art von logischer Beschreibung) → BDI-beschreibbar
  - Stereotyp «persistent» für persistente Daten (nach Neustart / Migration noch vorhanden)
- Beschreibung von BDI-Agenten
  - *belief*, *desire*, *intention*, *goals* als Instanzvariablen (Typ wff)
- Andere Strategien
  - permanent-goals, actual-goals als wff-Felder bei rein zielorientierter Semantik
  - auch denkbar: verschiedene Strategien (Konzept BDI, dann Implementierung auf Java-basierter Agentenplattform)

Aktionen und Methoden

- Aktionstypen
  - Stereotyp «proactive»: ohne externen Auslöser (interne Logik, Timer, auch über eigene Sensoren);
  - Stereotyp «reactive»: ausgelöst durch externe Nachricht
  - Beschreibung wie eine Methode: mit Signatur, Sichtbarkeitsattribut, Aktionsname, Parameterliste
  - Semantik wieder definieren über Vorbedingungen, Nachbedingungen, Effekte, Invarianten
- Methoden
  - genau wie in UML, ggf. mit Vor- und Nachbedingungen, Effekten, Invarianten
- Fähigkeiten (Capabilities)
  - informell oder mit Klassendiagramm (z.B. für FIPA-Dienstbeschreibungen)

Senden/Empfangen kommunikativer Akte

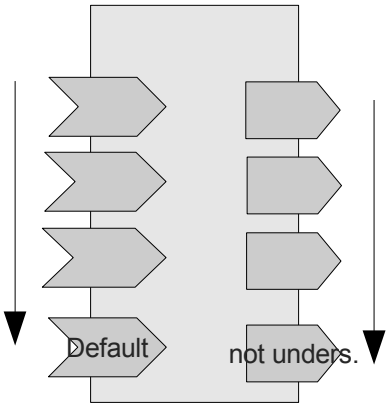
- Kommun. Akt umfasst Typ der Nachricht, Sender, Empfänger, Inhalt
- Notation
  - CA-1 / prot1: Akt der Klasse CA-1 findet statt als Teil des Protokolls prot1; auch notiert als: prot1[CA-1]
  - CA-1 / prot1 oder prot1[CA-1]: konkrete Instanz
  - Weglassen von „/ Protokoll“, wenn CA unabhängig vom Protokoll
- Default: unspezifizierter CA
- not-understood: Nachricht wurde nicht verstanden
- Klasse eignet sich zur Abstraktion von festen Werten
  - erlaubt generische Formulierung



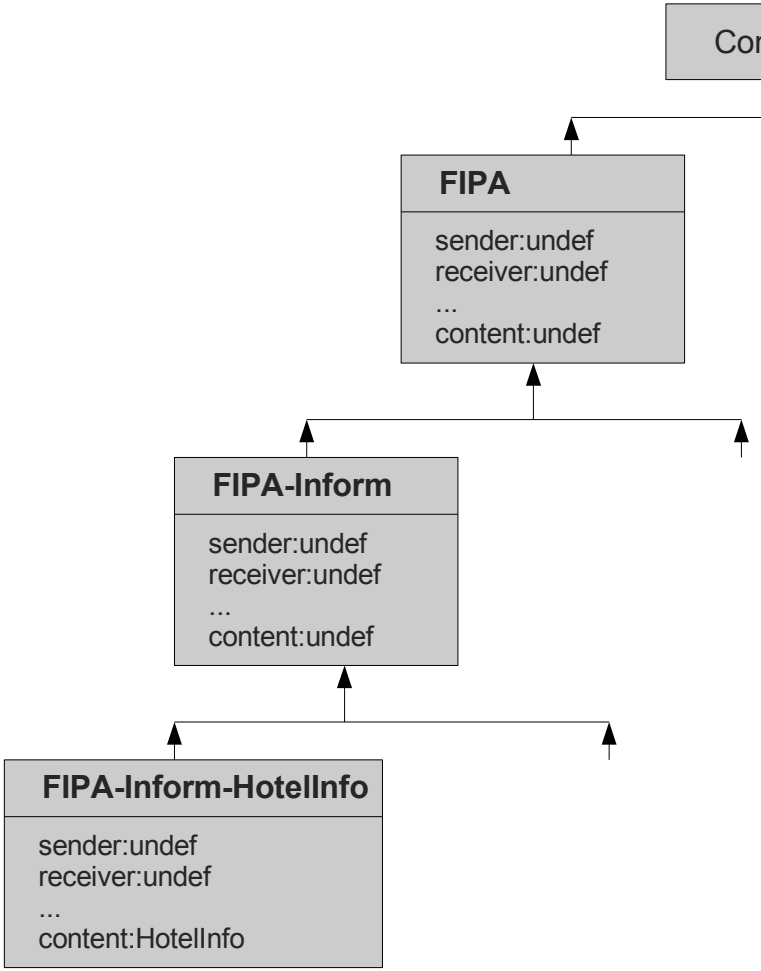
Kommunikative Akte

- Vergleich von CAs
  - CAs werden als Klassen modelliert
    - ohne Methoden
    - nur Instanzvariablen
  - daraus Aussagen wie
    - Klasse CA1 passt zu CA2 (Subtyp)
    - Instanz CA1 passt zu CA1
    - Instanz CA1 passt zu Instanz CA2

Matching erfolgt in der Reihenfolge von oben nach unten



Kommunikative Akte



Klassen ohne Methoden

Definition freier Variablen in CAs
Wert **undef**: kann von Instanz frei belegt werden

Bei Matching mit Protokoll:
protocol[CA] passt auf protocol'[CA'], wenn CA auf CA' passt und protocol=protocol' ist

Matching-Regeln

- Der Eingabe-Akt CA passt auf („matches“) die eintreffende Nachricht CA' genau dann, wenn gilt:

CA ist eine Klasse

- CA' muss Instanz von CA sein
- CA' muss von CA abstammen (also Subklasse, ggf. mehrere Generationen)

CA ist eine Instanz einer Klasse

- CA' ist Instanz der Klasse von CA
- CA.Feld passt auf CA'.Feld bezüglich jedes Feldes von CA,

wobei hierfür gilt: CA.Feld passt auf CA'.Feld, wenn
- CA.Feld mit undef belegt ist
- CA.Feld gleich CA'.Feld ist, wenn CA.Feld nicht undef ist und der Typ von Feld ein Basistyp ist
- ansonsten CA.Feld und CA'.Feld Instanzen derselben Klasse C sind und hierbei CA.Feld.CFeld auf jedes CA'.Feld.CFeld passt (für jedes Feld von C)

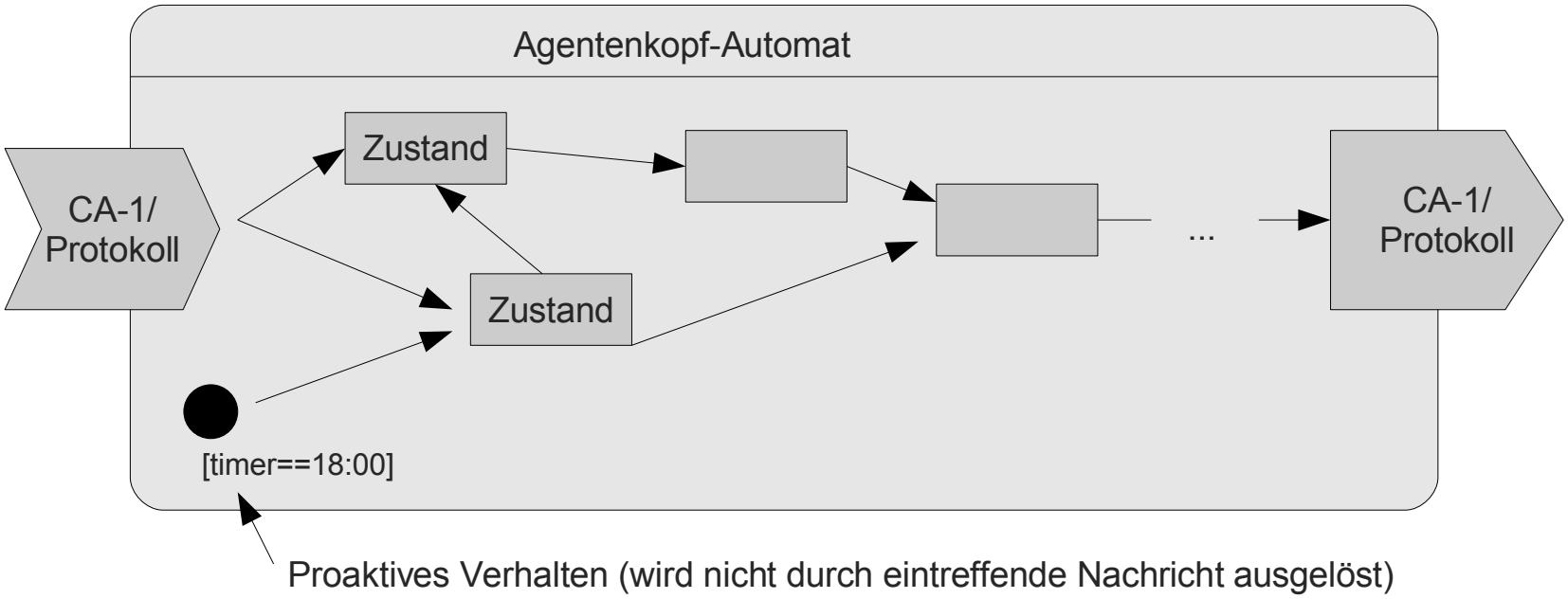
Analog für die ausgehenden CAs und Nachrichten

Das Verhalten

- definiert im „Kopf“ (agent head) als Zustandsautomat
  - definiert reaktives und proaktives Verhalten
- Möglichkeiten der Spezifizierung durch UML
  - Sequenzdiagramme, Kollaborationsdiagramme (auf Ebene der Objekte)
    - konkreter, kann leichter in Implementierung umgesetzt werden
  - Zustands- und Aktivitätsdiagramme
    - abstrakter, weniger Implementationsdetails

Das Verhalten

- Zustandsdiagramm für reaktives Verhalten
  - erweiterte Zustandsautomaten
  - Startzustand und Endzustand als CA



Vergleich

- Gaia
  - fokussiert auf Konzeptebene / Analyse; definiert eine Methodik
  - Konsequente Umsetzung der Sicht auf Agenten
    - Anwendung als Ensemble autonomer Einheiten entwerfen, sich nicht von traditionellen Methoden die Sicht einengen lassen
    - Idee einer „Stellenausschreibung“ für Agenten; Posten definieren und mit Agenten „besetzen“
- AgentUML
  - Erweiterung traditioneller Technik
  - Beruht auf UML 1.x; teilweise in UML 2.x aufgegangen
  - Detaillierter Entwurf, Protokollebene, Implementierungsnähe
  - Für spätere Analysezwecke/Dokumentation gut geeignet, aber nicht für frühen Entwurf

Weitere Ansätze

- Tropos (Paolo Bresciani)
  - Visuelle Modellierung
  - Vollständige Abdeckung des Entwicklungsprozesses, angefangen bei frühen Anforderungen bis zum Detailentwurf
- Prometheus (Lin Padgham und Michael Winikoff)
  - Grafischer Editor
  - Drei Phasen des Entwurfs
    - Systemspezifikation: Identifikation von Aktoren, Zielen, Arbeitsschritte des Szenarios
    - Abstrakter Entwurf (high-level): Definition von Agententypen; architektureller Überblick
    - Detailentwurf: Fähigkeiten der Agenten, Pläne, Daten usw.
  - Verbindung zum Agentensystem Jack (gleiche Quelle)