Projektgruppe

Bearbeiter: **Othmane Rhandor**

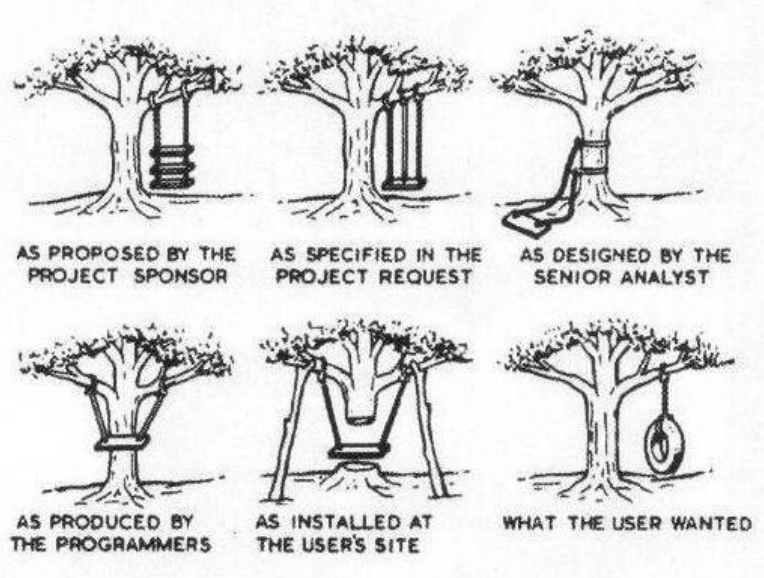
Automatische Modellgenerierung aus Systemspezifikationen

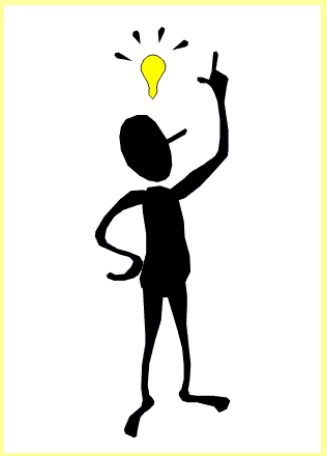
Inhalt

- Motivation
- Ansätze
- MOF und EMF
- Zusammenfassung

Motivation

- Systemspezifikationen (Anforderungen) in natürlicher Sprachen (NL)
  - Teilweise subjektiv → viel Interpretationsraum
  - Erschwert die Kommunikation

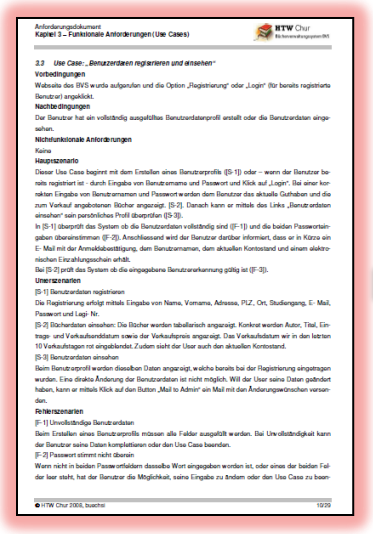




→ Verzögern den gesamten Entwicklungsprozess

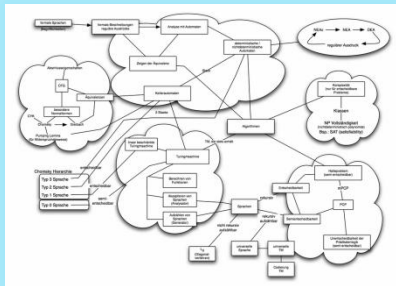
Motivation

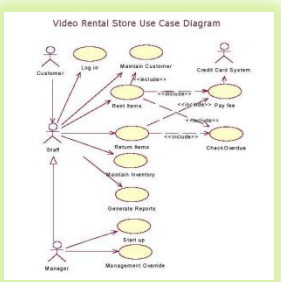
Ziel: Automatische Modellgenerierung aus Anforderungsdokument



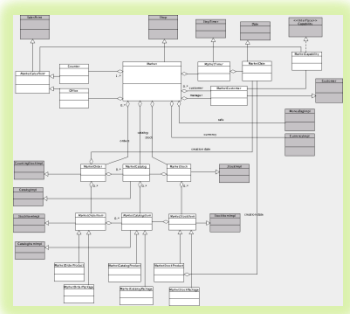
Anforderungen in NL

Werkzeug / Methodik

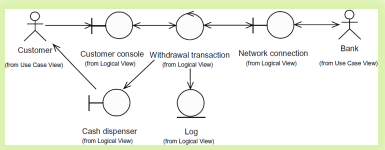




Use Case Diagramm



Klassen-Diagramm



Analyse-Diagramm

Inhalt

- Motivation
- Ansätze
- MOF und EMF
- Zusammenfassung

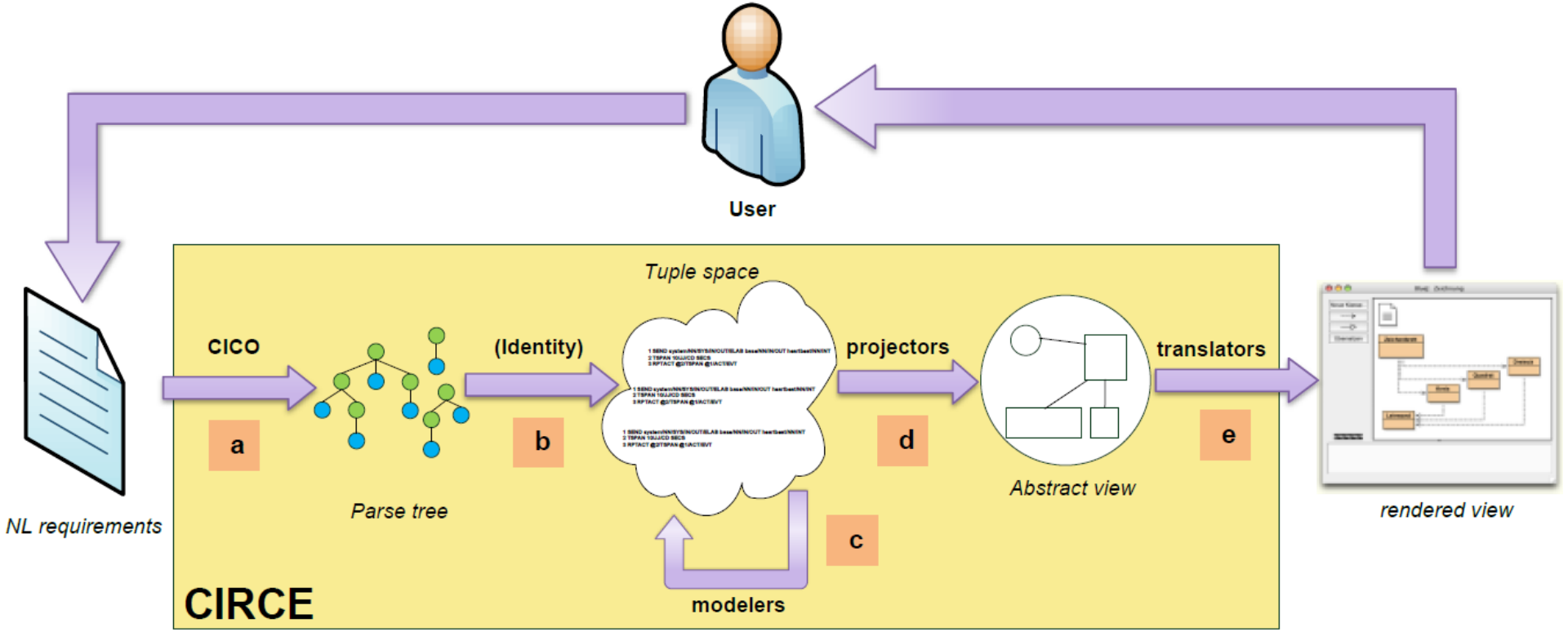
Ansätze

- **Ansatz 1:** **C**ooperative **I**nteractive **R**equirements-**C**entered **E**nvironment (CIRCE)
  - Entwickelt von *Vincenzo Gervasi* (Uni Pisa, Intalien)
  - Webbasierter Werkzeug
- **Ansatz 2:** Use Case driven Development Assistant (UCDA)
  - Entwickelt von *Dong Liu* (Uni Calgary, Canada)
  - Plug-In für IBM Rational Rose

→ **Methodik!**

Ansatz: CIRCE

- Architektur:



Definitionen

- Anforderungsdokument D = <G, F, R>
  - **G** = Glossar
    - Domainspezifische Begriffe und Entitäten
    - Synonyme und Metonymien
    - Begriffe markiert (Tags)
  - **F** = Definitionen
    - Kürzel der Anforderungen
    - Bringt Text in kanonischer Form
  - **R** = Anforderungen
    - Erwartetes Systemverhalten
    - Dient als Input für den Parser

Beispiele

- *Requirement :: **R***

  Wenn das Wasser kocht, dann muss der Wasserkocher innerhalb 5 sec ein Signal zum Abschalten ausgelösen.

- *Definitions :: **F***

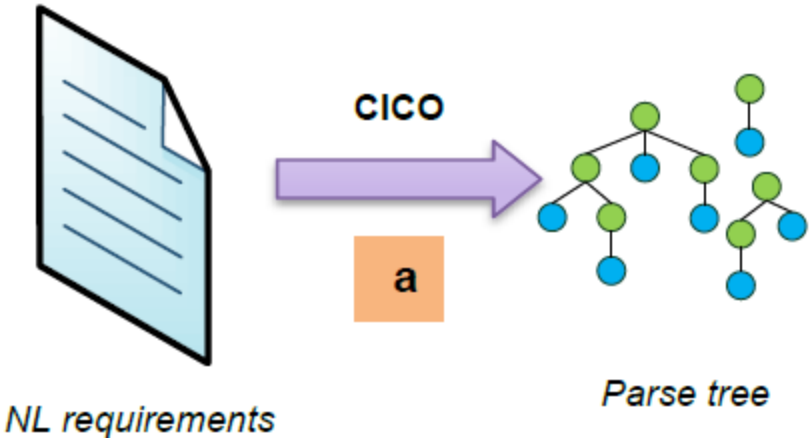
  SCHALTE x AB → SENDE EIN ABCHALTE SIGNAL AN x

- *Glossars :: **G***

  - Formal: $\{(\tau, \{\tau_i\})\}$
    - $\boldsymbol{\tau}$: Menge der Begriff
    - $\boldsymbol{\tau_i}$: Synonyme
  - Bsp.: G = {(*System/IN/OUT*, {*Wasserkocher, User*})}

(a) CICO

- ***Input***: Dokument in kanonischer Form



- Eigenschaften:
  - Einfacher Domain-basierter Parser
  - Fuzzy-Matching (templates) anstatt Sprachgrammatik
  - Model-Action-Substitution (MAS)
  - Benutzung Optimierungsheuristiken

(a) CICO – Vorgehensweise

- Morphosyntaktische Analyse – *part-of-speech* (POS)

  →Benutzte Variante: Penn Treebank tagset

  →**Ergebnis**: $s = \langle w_1, \ldots, w_n \rangle$

- Model-Action-Substitution Regeln (MAS): $\rho = (\mu, \alpha, \sigma)$

  → Regeln sind frei programmierbar

- Matching-Funktion *match(ρ, s)*

(a) CICO – Beispiel

- $\rho = (\mu, \alpha, \sigma)$

  μ = snd/OUT SEND data/INF TO recv/IN

  α = SEND $snd $recv $data

  σ = $ID/ACT/EVT

- Init text = the systemshall send a heartbeat to the base

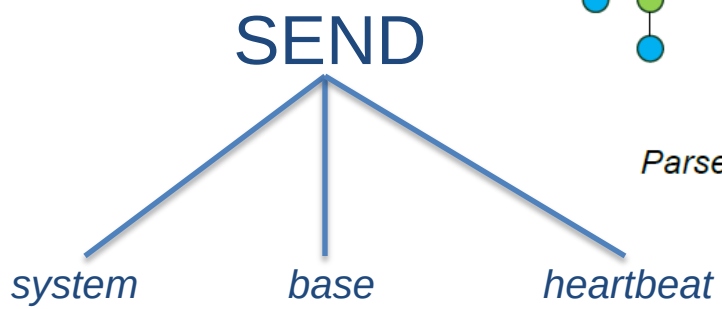
- s = the/DT system/NN/SYS/IN/OUT/ELAB shall/MD send/VB a/DT heartbeat/NN/INF to/TO the/DT base/NN/IN/OUT

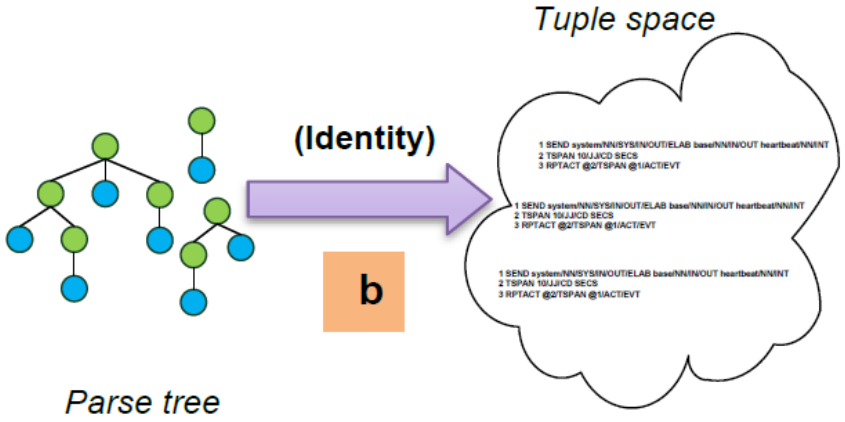
match(ρ, s) = <{system, base}, {send}, {heartbeat}, {to}, {system, base}>

snd SEND data TO rcv

(a) Parsen durch CICO

- *α = SEND $snd $recv $data*



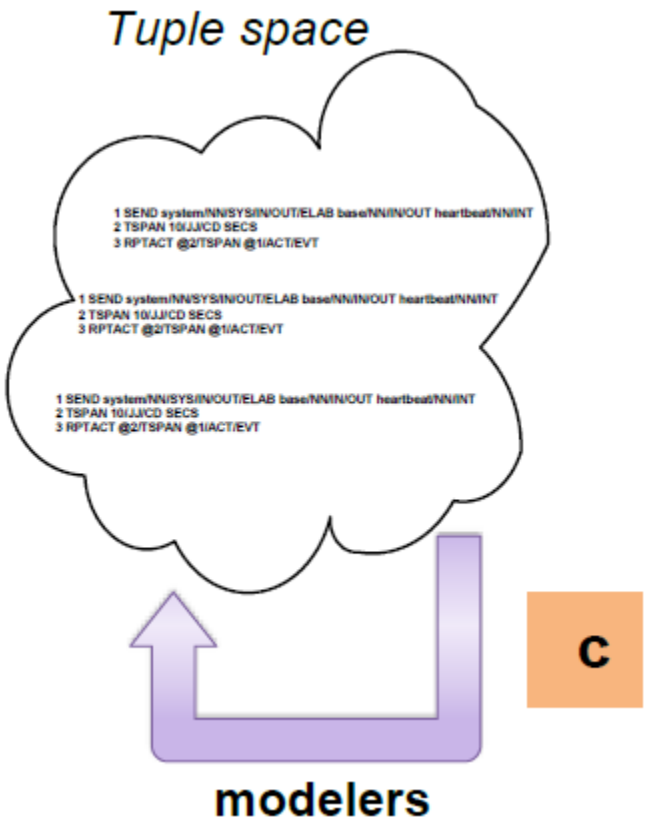


- *σ = $ID/ACT/EVT*
- **Codierter Tupel:**

→1 SEND system/NN/SYS/IN/OUT/ELAB base/NN/IN/OUT heartbeat/NN/INT

(c) Modelers

- Erzeugen neue Tupel
- Intensionale Wissen über das System
- Modelers Strunktur: *M = <i, o, f>*
  - *i = input tupels*
  - *o = output tupels*
  - *f = Logik als Funktion*
- *Modelers Typen Beispiele:*
  - *Abstraction Modelers*
  - *Intensional Modelers*
  - *Validierungsmodelers*



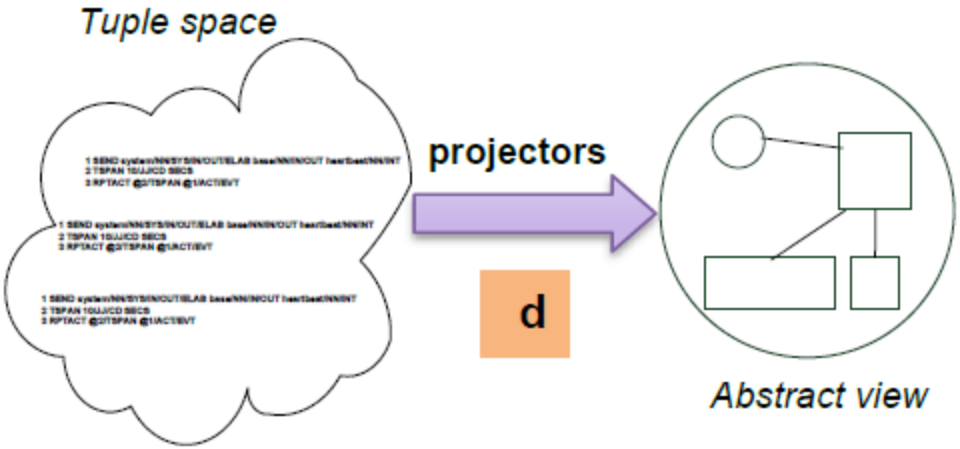
(c) Modelers – Beispiel

Abstraction Modeler:

$$m_{iface} = \langle \{SEND, RECEIVE\}, \{IIFACE, OIFACE\}, f \rangle$$

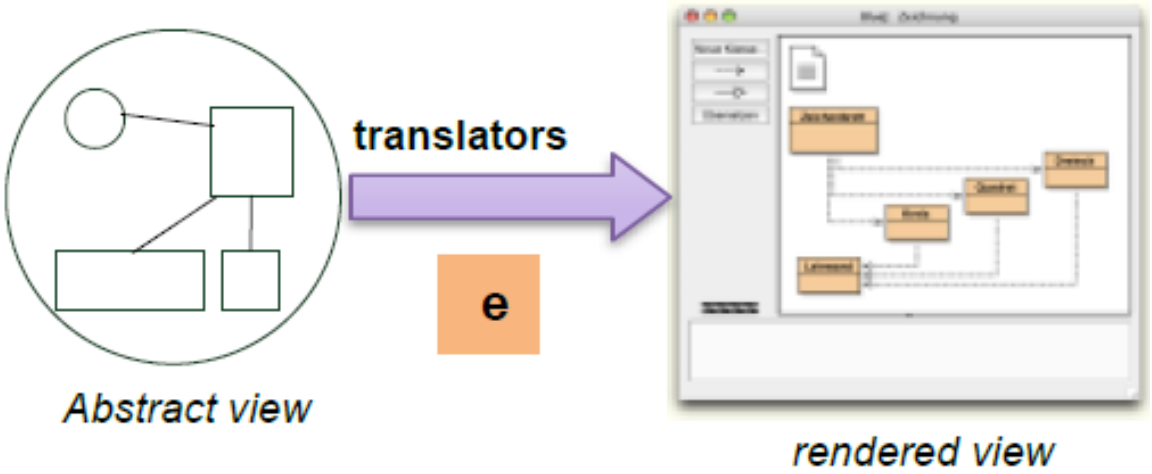
- ***SEND, RECEIVE**:* Kommunikationsaktionen
- ***IIFACE, OIFACE**:* System Schnittstelle Tupeltyp
- Funktion ***f**:* beschreibt die Logik
  - *iiface(src, dst) ← send(src, dst, data) ^ ┐system(src) ^ system(dst)*
  - *iiface(src, dst) ← receive(dst, src, data) ^ ┐system(src) ^ system(dst)*
  - *oiface(src, dst) ← send(src, dst, data) ^ system(src) ^ ┐system(dst)*
  - *oiface(src, dst) ← receive(dst, src, data) ^ system(src) ^ ┐system(dst)*

(d) Projectors



- Native Meta-Model
- Typische Tags:
  - /ENTITY: physikalische und logische Entitäten
  - /ATTR: Attribute der Entitäten
  - Beispiel: Base/ENTITY, System/ENTITY

(e) Traslators



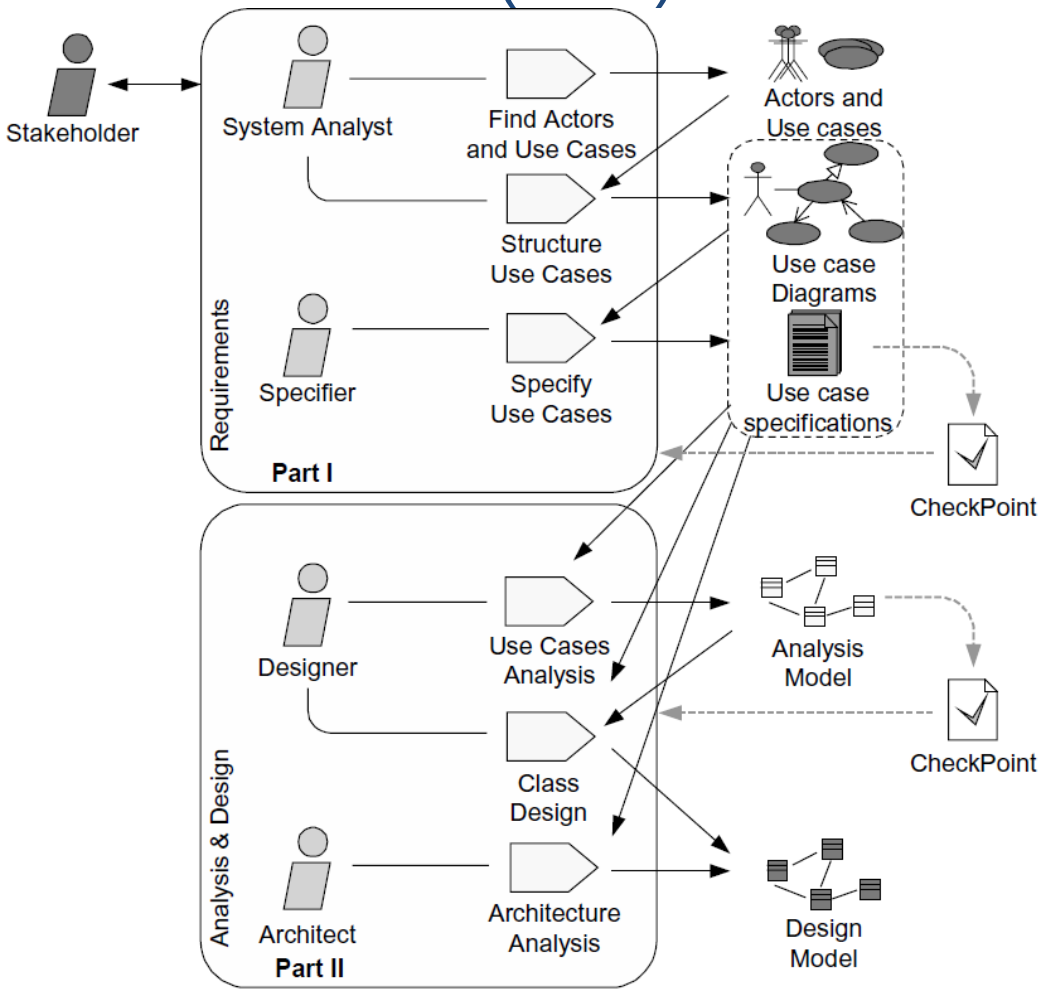
- Abbildung der abstrakten Sicht
- UML Modelle
- Exportieren als XMI

Inhalt

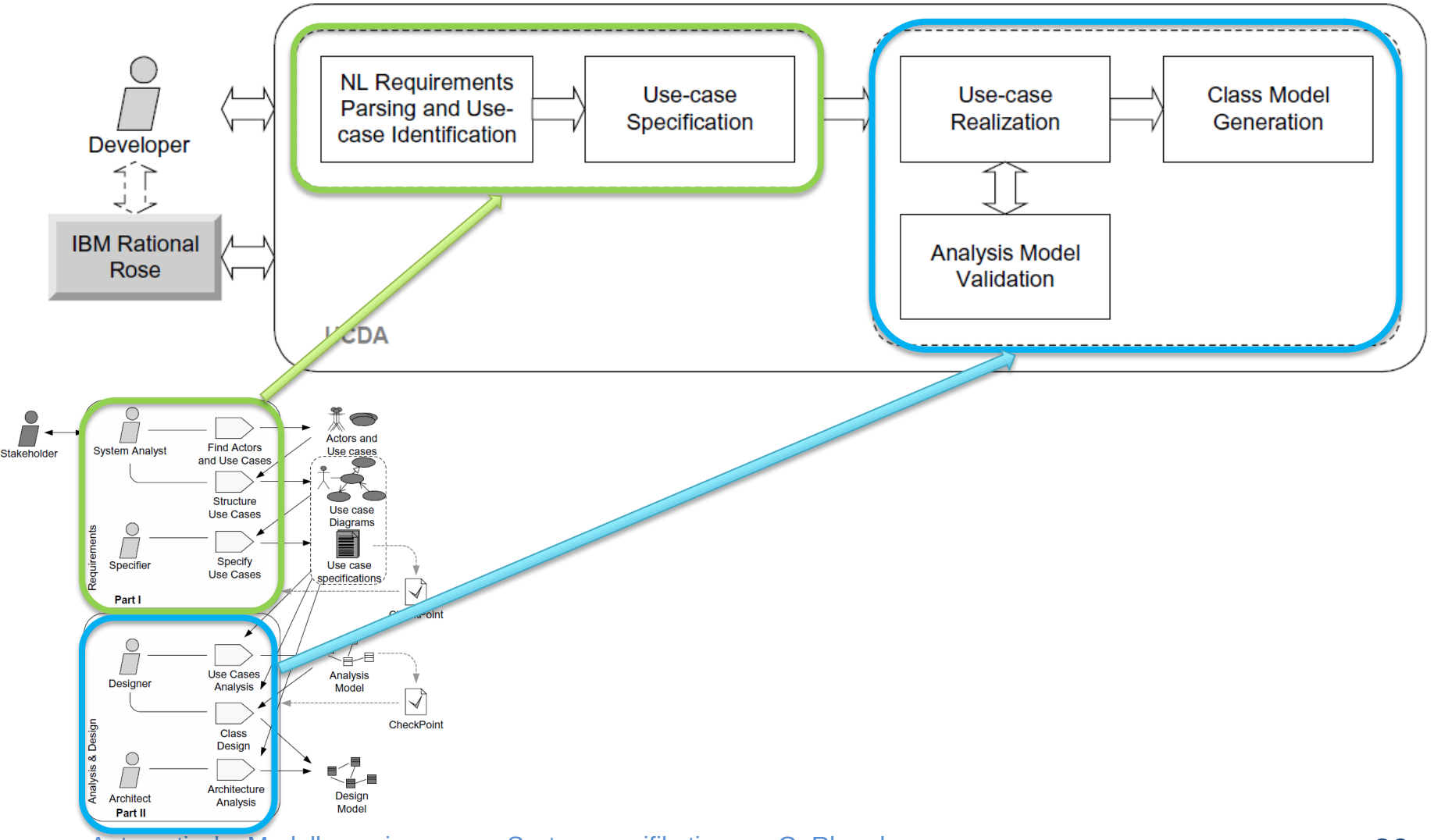
- Motivation
- Ansätze
  - CIRCE
  - UCDA
- MOF und EMF
- Zusammenfassung

Ansatz: UCDA

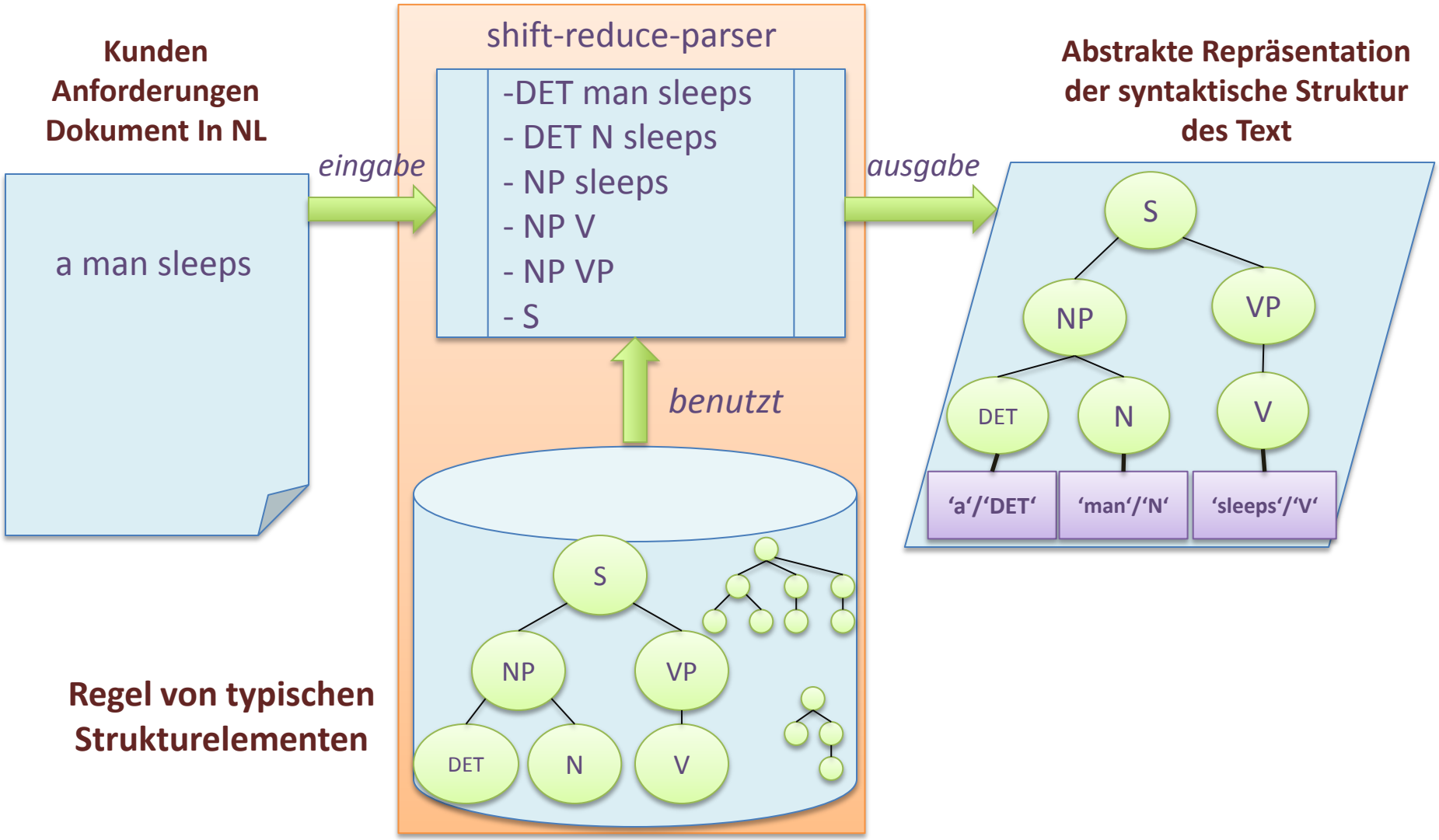
Rational Unified Process (RUP):



UCDA – Architektur



NL parsen


Kunden Anforderungen Dokument In NL
a man sleeps
eingabe
shift-reduce-parser
-DET man sleeps
- DET N sleeps
- NP sleeps
- NP V
- NP VP
- S
benutzt
ausgabe
Abstrakte Repräsentation der syntaktische Struktur des Text
S
NP
VP
DET
N
V
'a'/'DET'
'man'/'N'
'sleeps'/'V'
Regel von typischen Strukturelementen

Use Case identifizieren

- **Glossar als Vorrausetzung:**
  - Vordefinierte **Aktoren** und **Use Cases**
  - Wie: Durch Verwendung von Heuristiken, wie z.B.:
    - Wer stellt zur Verfügung, benutzt oder löscht Informationen?
    - Wer benutzt die Funktionalität?
- **Aktoren** Kandidaten: Nomen (Substantive)

  → *Aber*: Aktoren die nicht in Glossar werden entfernt
- **Use Cases** Kandidaten: Verb Ausdrücke (Phrasen)

Use Cases identifizieren - Beispiel

Anforderungen einer Bankautomat.

„Der Automat bedient maximal einen Kunden zu jeder Zeitpunkt. Ein Kunde startet eine Sitzung sobald er Bankkarte einsteckt. Der Kunde wird dann in der Lage sein eine Transaktion durchzuführen.“

→ **Aktor** = „Kunde“

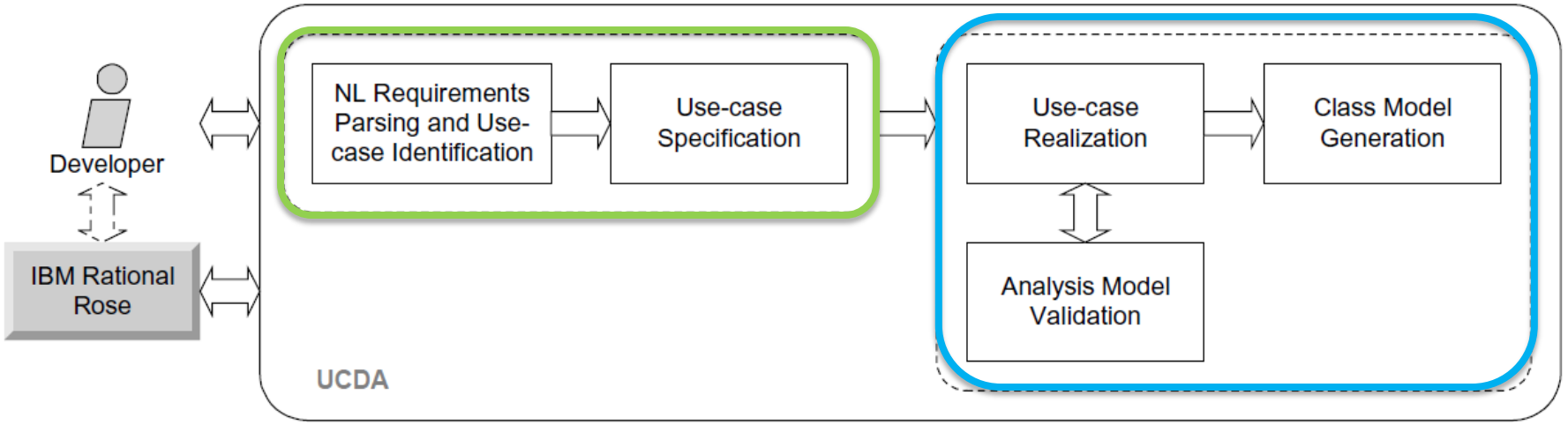
→ **Use Cases**: „Sitzung starten“ und „Transaktion durchführen“

Use Cases spezifizieren

- Basierend auf ein „Template“
  - Use Case name
  - event flow
  - pre-condition
  - Etc.
- Schema zum automatisieren der Use Cases Spezifikation
  - Basic
  - If-then
  - Do-until
  - Con-Noc

Use Cases realisieren, Analyse- und Klassenmodell generieren

Input: Aktoren, Use Cases und Use Case Spezifikation

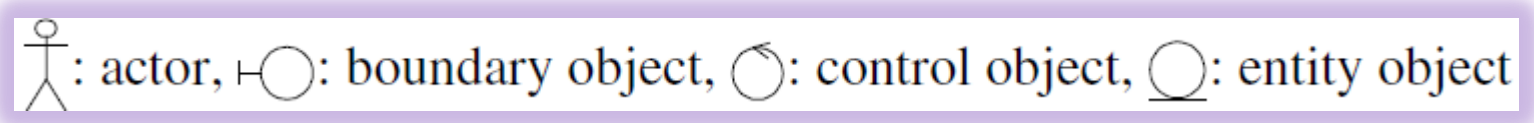


Output: Aktoren, Use Cases und Use Case Spezifikation

Use Cases realisieren, Analyse- und Klassenmodell generieren (2)

Use Cases Realisierung:

- Struktur der Handlungsaussagen
- Verhaltenstypen des Systems
- Stereotypen der Objekte



→Potentielle Instanzen einer Klasse: Entitäten

- Vererbung und Komposition

Use Cases realisieren, Analyse- und Klassenmodell generieren (3)

- Für jeden Use Case:
  - Analyseklassen, Stereotype → Robustheitsdiagramm
  - Das Systemverhalten zerlegen → Kollaborationsdiagramm
- Für die Analyseklassen
  - Verantwortlichkeiten beschreiben
  - Verbindungen beschreiben
  - Die Vererbungsverbindungen erzeugen

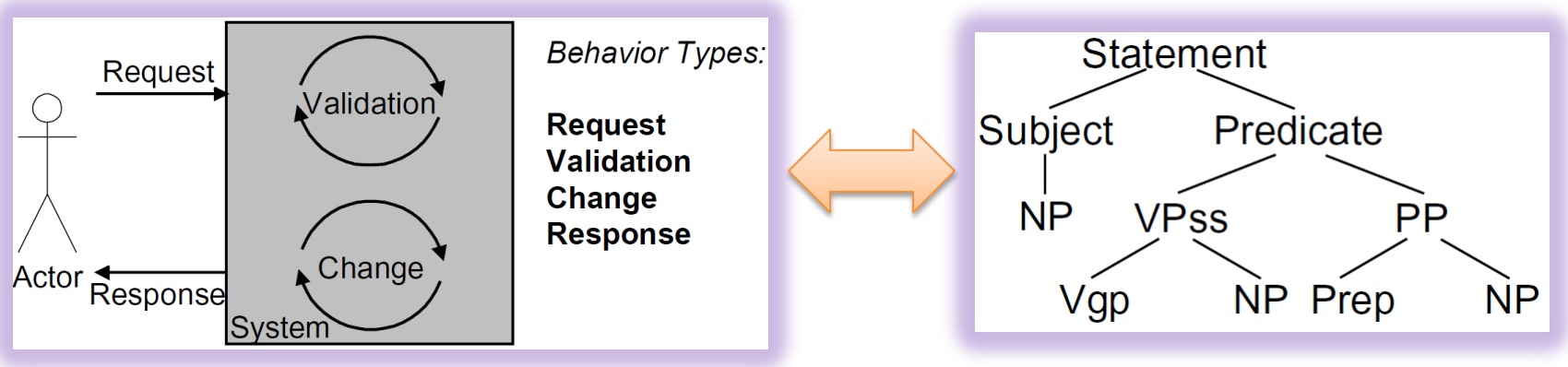
→ **Ja, aber Wie**: Anwenden von Regel!

Use Cases realisieren, Analyse- und Klassenmodell generieren (4)

- Insgesamt sind es 17 Regeln

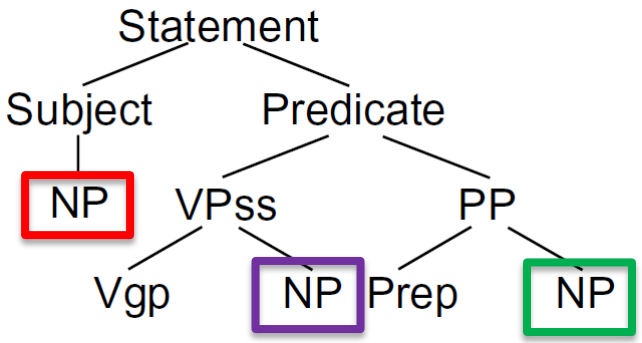
  →Identifizierung von Objekte und Nachrichten

- Basierend auf den Satzstruktur und Systemverhalten



Use Cases realisieren, Analyse- und Klassenmodell generieren (5)

Beispiel: Transitiver Satzstruktur

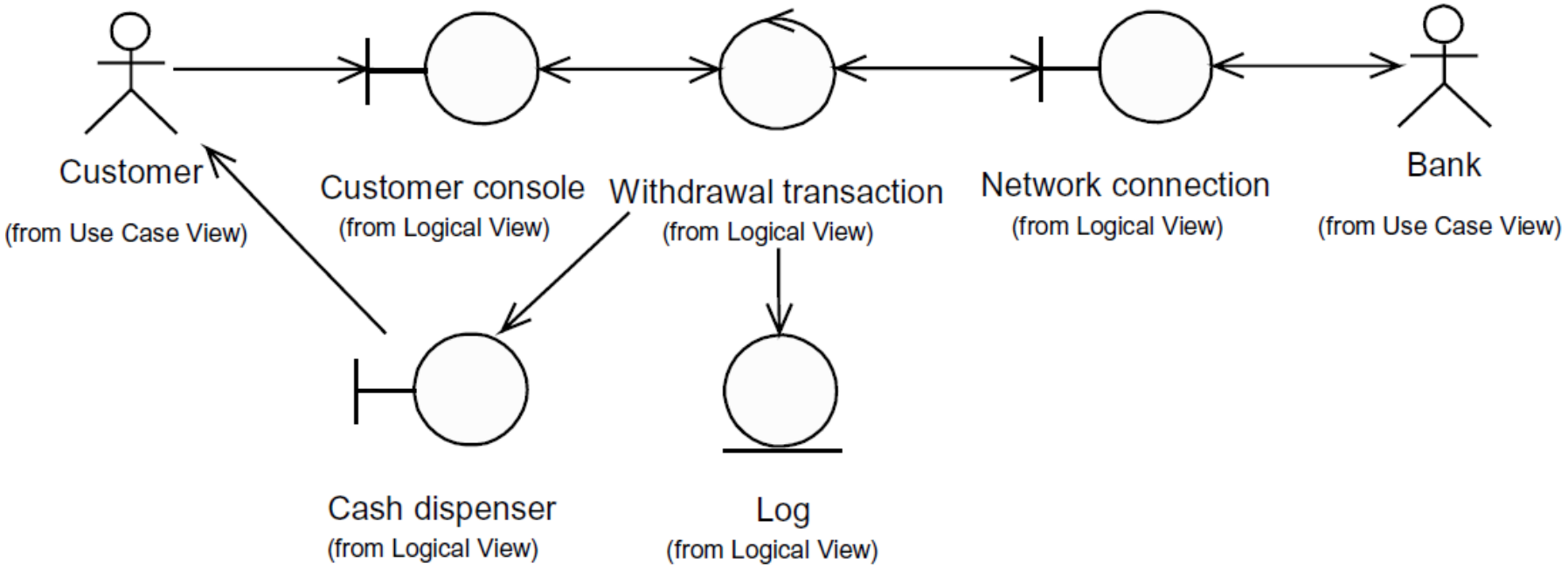


Regel:

*”If the structure of a statement is transitive, and Subject/NP//Noun(head) is an **actor**, then this statement is corresponding to the **Request** behavior type and Predicate/PP/NP//Noun(head) is a **boundary** object if it exists in the glossary, and Predicate/VPss/NP/Noun(head) is an **entity** object if it exists in the glossary.“*

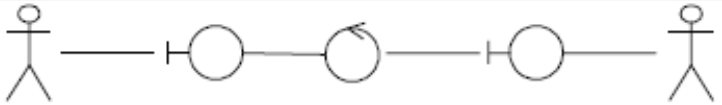
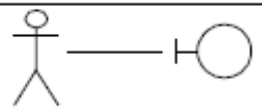
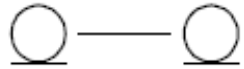
*Der **Benutzer bestätigt die Transaktion** auf der **Konsole**.*

Beispiel: Robustheitsdiagramm



Analysemodell Validierungsregeln

Beispiel für Analyse Regel:

Case	*Validation*	*Suggestion*
	*Not allowed.*	
	*Allowed*	
	*Not allowed.*	

UCDA – Zusammenfassung

- Basierend auf eine systematische Methodik – RUP
- Architektur besteht aus zwei Teile
  - Teil 1: Requirments aus NL parsen, Use Cases identifizieren und spezifizieren
  - Teil 2: Use Cases realisieren, Klassenmodelle generieren und validieren
- Umsetzung als Plug-In für die IBM Rational Rose Entwicklungsumgebung

  → **XMI Export interssant für die PG!**

Inhalt

- Motivation
- Ansätze
- MOF und EMF
- Zusammenfassung

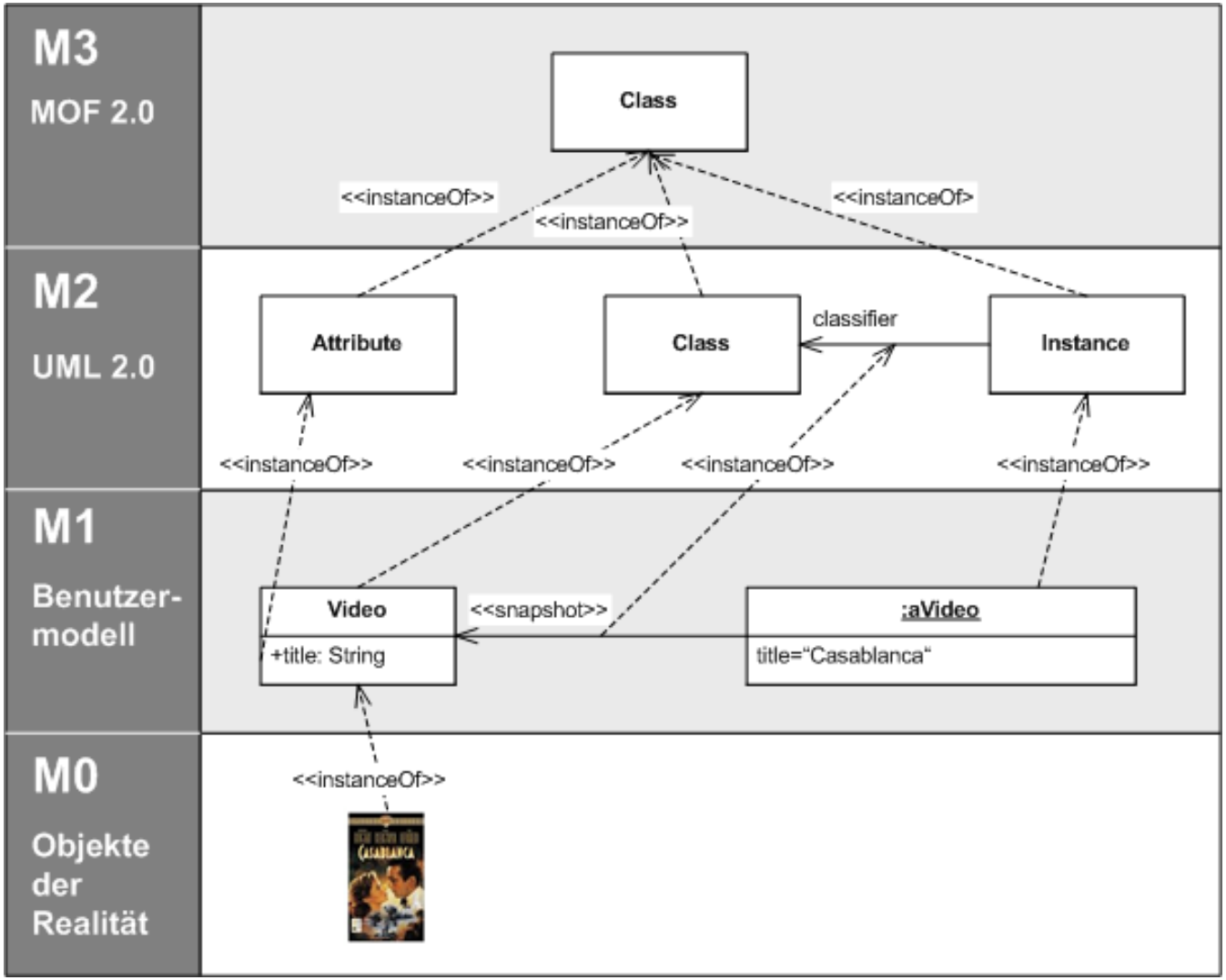
Meta-Object Facility (MOF)

- Object Management Group™ (OMG™)
- Meta-Meta Sprache
  - Meta Modellierung Framework
  - Domain spezifische Sprachen

    → UML Modellierungssprache
- **XMI-Format als Metadaten Austauschformat**

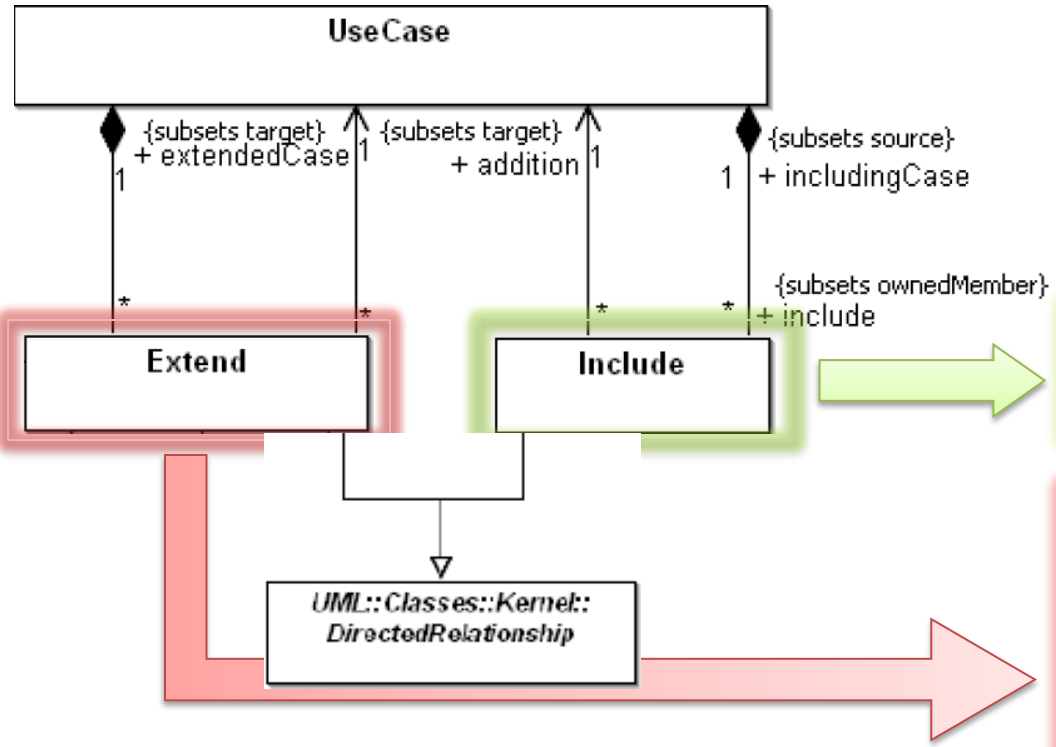
  → Modelle die sich mit MOF ausdrücken lassen (z.B.: UML)

MOF und EMF

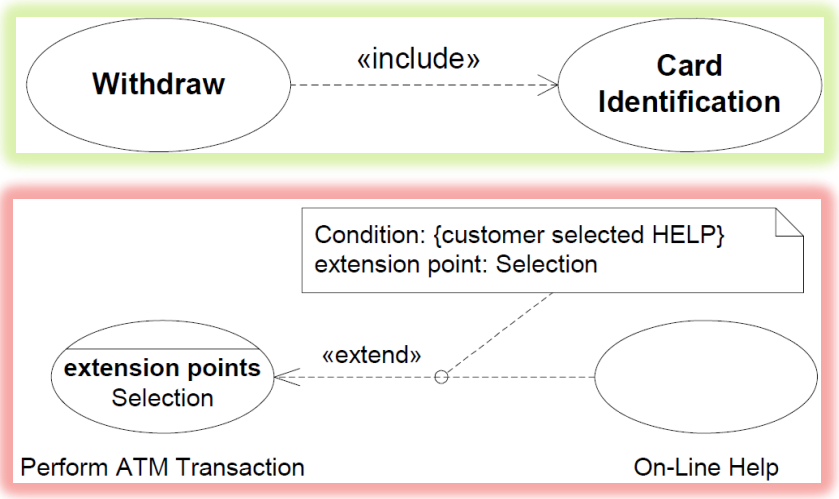

M3
MOF 2.0
Class
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
M2
UML 2.0
Attribute
Class
classifier
Instance
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
M1
Benutzer-modell
Video
+title: String
<<snapshot>>
:aVideo
title="Casablanca"
M0
Objekte der Realität
<<instanceOf>>

MOF Beispiel: Use Case Diagramm

UML Meta-Modell



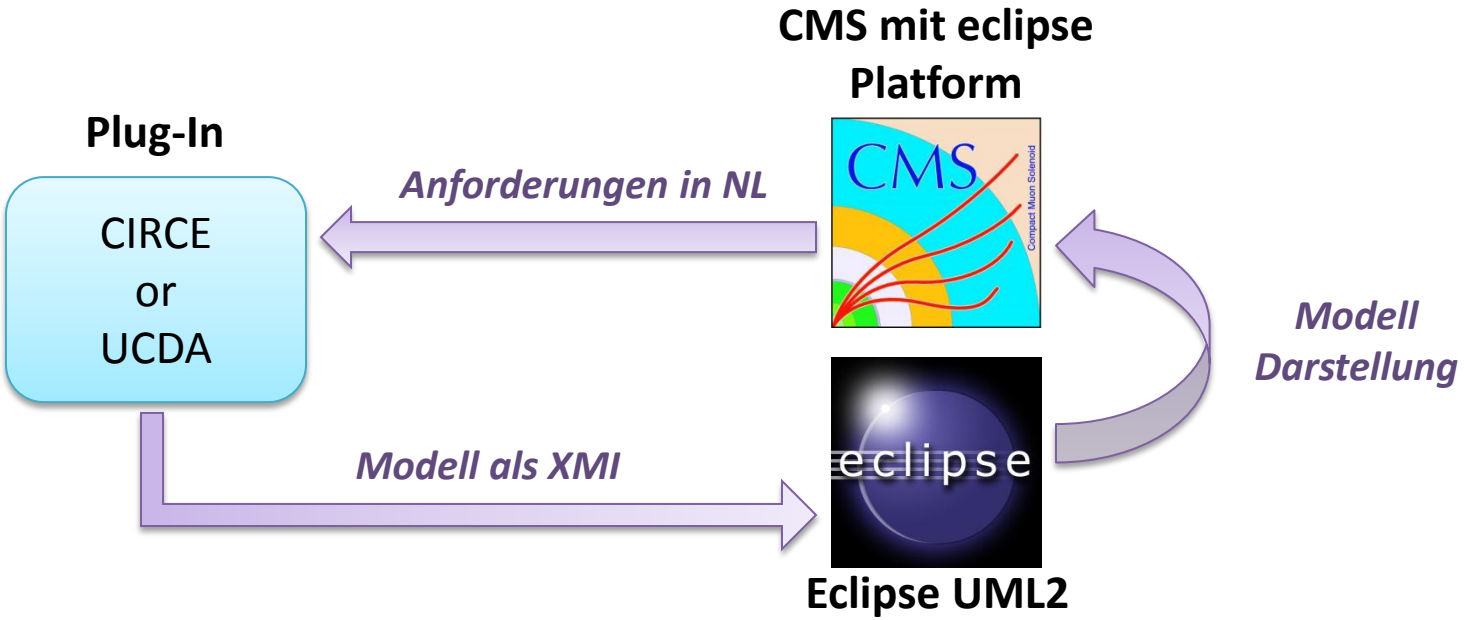
Modell Beispiel



MOF und EMF

Eclipse Modeling Framework (EMF)

- Meta-Meta Modell für eclipse Platform
- XMI-Format als Metadaten Austauschformat
- **Für die PG:** "*Eclipse UML2*" als Umsetzung für OMG UML 2.x



Inhalt

- Motivation
- Ansätze
- MOF und EMF
- Zusammenfassung

Zusammenfassung

- Problematik:
  - Anforderungen in NL formalisieren (Semi-formale Modelle)
- Vergleich der Lösungsprinzipien

Vergleichspunkt	CIRCE	UCDA
Modelltypen	Statische + Dynamische	Statische
Parsen	Domain-basiert	Sprachgrammatik-basiert
Integration	Nicht Möglich (in Bearbeitung)	Möglich (als Plug-In)
Portierbarkeit der Modell	XMI	XMI

Zusammenfassung (2)

- Modelle in CMS Integrieren
  - XMI als Austauschformat
  - Benutzer Integration für die Modellanalyse
  - EMF als Einsatzmöglichkeit
    - →Eclipse UML2 als Umsetzung für UML
- Eclipse als Platform für CMS

Vielen Dank für eure Aufmerksamkeit!