Modul OMSI-2
im SoSe 2010

Objektorientierte Simulation mit ODEMx

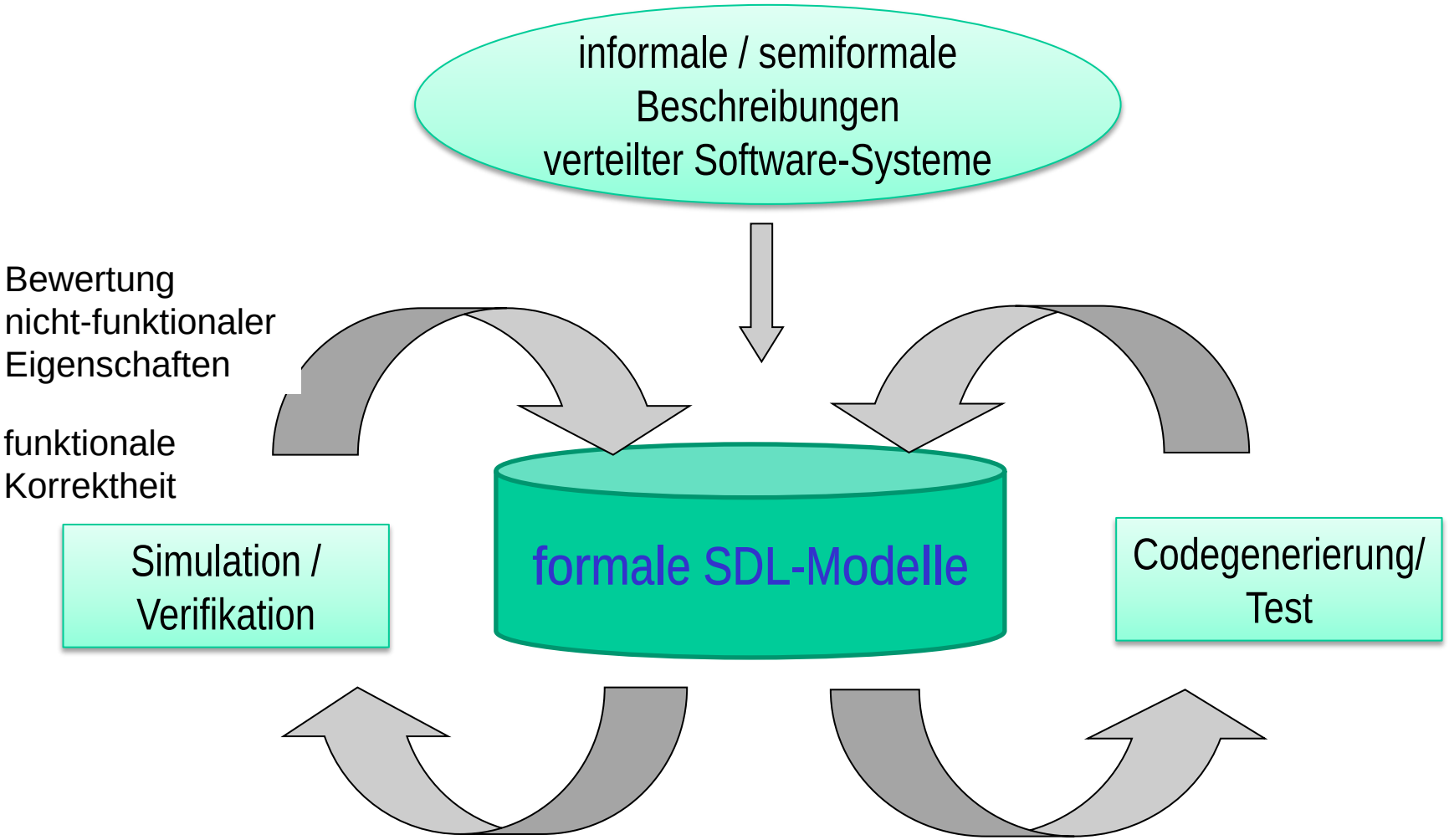
Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dipl.-Inf. Ingmar Eveslage
Dipl.-Inf. Andreas Blunk

fischer|ahrens|eveslage|blunk@informatik.hu-berlin.de

6. SDL

1. Grundphilosophie
2. ITU-Standard Z.100
3. Werkzeuge
4. SDL-Grundkonzepte
5. Musterbeispiel (in UML-Strukturen)
6. Struktur- und Verhaltensbeschreibung in SDL-RT

SDL= Specification and Description Language



Zur Geschichte

Einstieg der HU Berlin

- SDL (Specification and Description Language), Entwicklungsbeginn 1975
  standardisiert von der ITU (CCITT), Genf
- verschiedene Versionen (im Abstand von 4 Jahren) Z.100
  mit stabilen Kernkonzepten (asynchron kommunizierender Automaten)
- seit 1988 mit Konzepten einer formalen (statischen u. dynamischen Semantik)
- seit 1996 mit objektorientierten Konzepten
- mit SDL-2000 (einheitliches Kalkül für formale Semantik: ASM (Gurjewich, MicroSoft), hierachische Automaten, OO-DT
- SDL-2000-Referenz-Compiler (ohne industriellen Nutzen)
- 2005 Wechsel der Kern-Entwickler-Community zur OMG (UML-2.0)
  Stabilisierung der UML-Sprache (insbesondere SDL als UML-Profil)

SDL→ Simula-67

(Simula-67-) Laufzeitsystem ereignisorientierte Prozess-Simulation — 1982

SDL→ C with Classes

(C-)Laufzeitsystem Vererbungsersatz ereignisorientierte Prozess-Simulation — 1988

ODEM in C++ — 1991

SDL/ASN.1→ C++

(C++-)Laufzeitsystem Zielcode-Generierung für Siemens-Middleware — 1998

SDL/UML/ASN.1→ C++

(C++-)Laufzeitsystem Simulation und Zielcode-Generierung — 2009

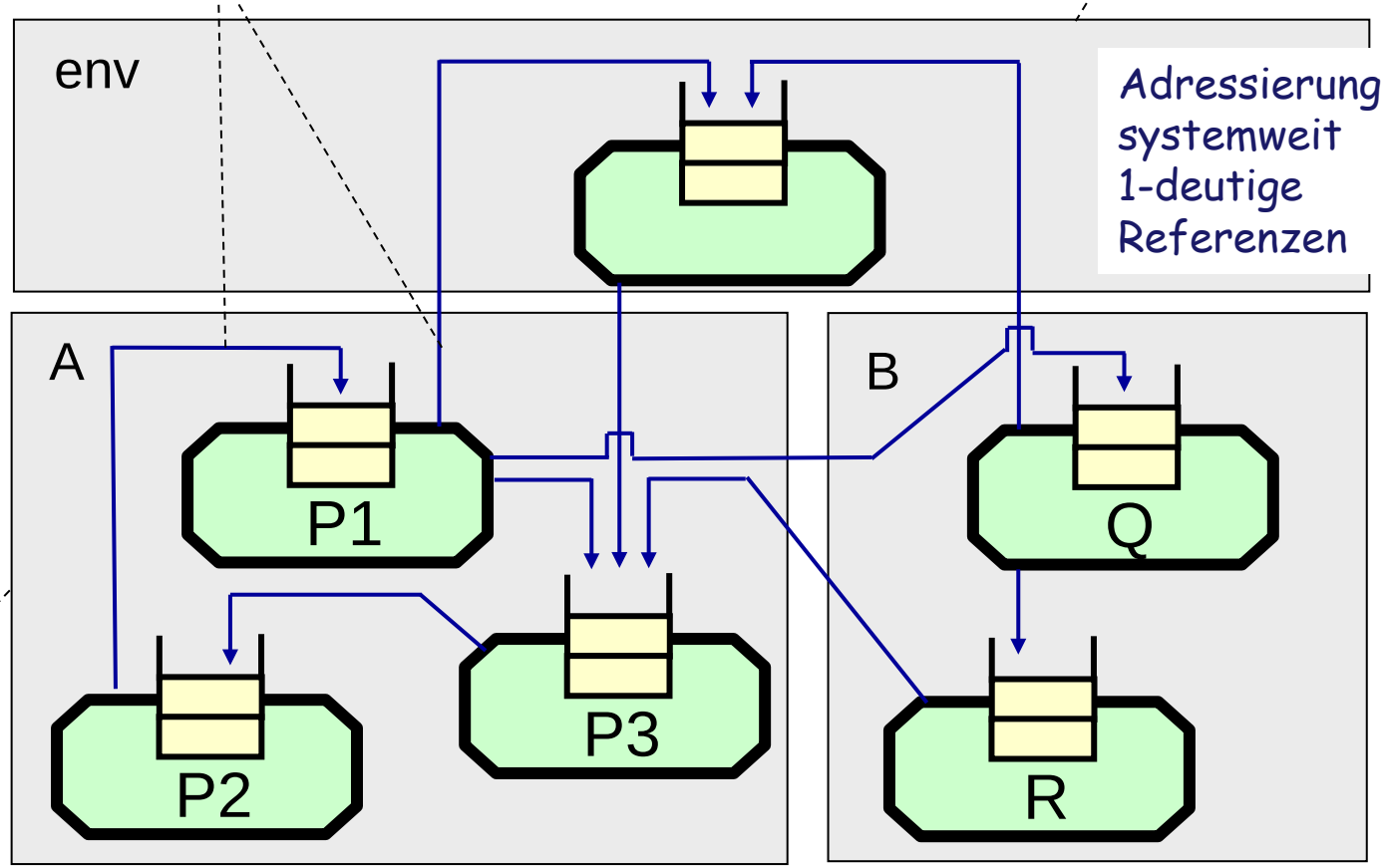
Laufzeitrepräsentation von SDL-Systemen

PType

potentieller Nachrichtenfluss über zeitlos bzw. zeitkonsumierende Übertragungskanäle

anonym strukturierte Systemumgebung

Agenten/ Prozesse können typbasiert oder repräsentanten-basiert definiert sein

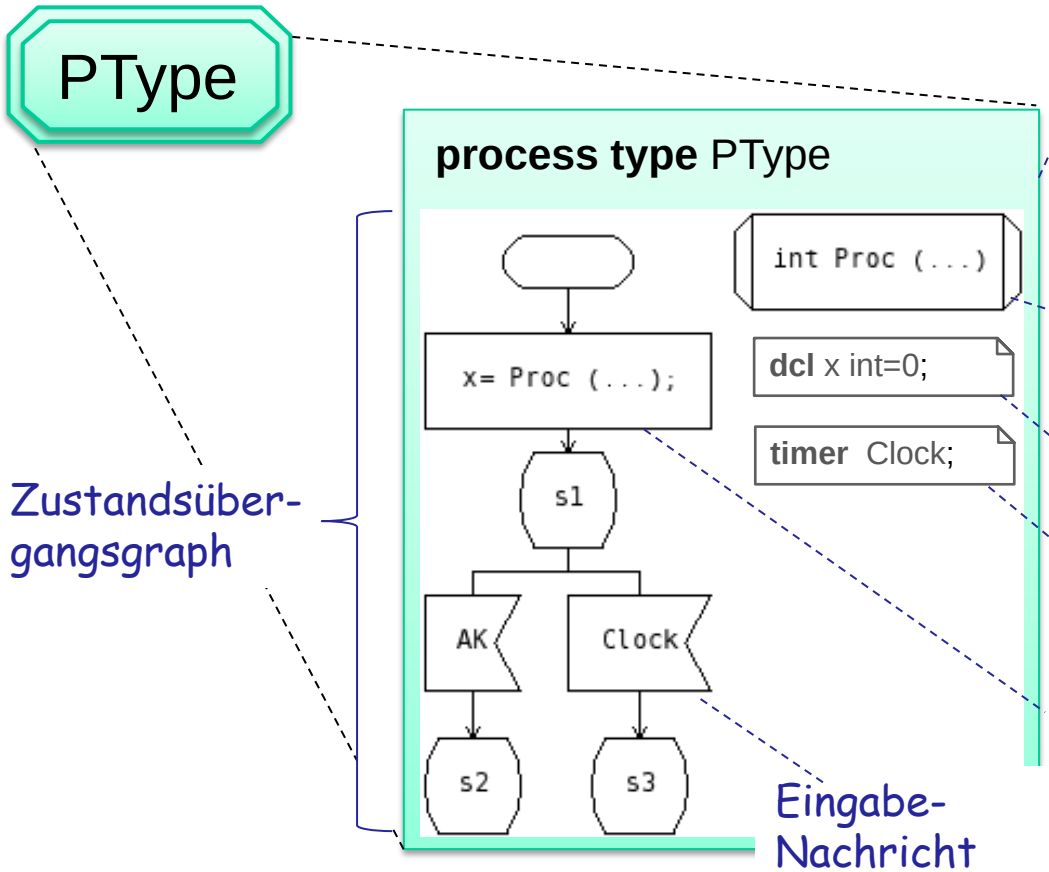


Strukturen von Agenten/ Prozessen (hier: A und B

asynchron kommunizierende Zustandsautomaten (Prozesse) mit impliziten (unbeschränkten) Nachrichtenempfangspuffer

Processe (Agenten): Typbeschreibung

Referenzsymbol des Prozesstyps PType



zugehöriges (per mouse click) Definitionsdiagramm

Referenzsymbol der lokalen Prozedur Proc (benötigt zugehöriges Definitionsdiagramm)

lokale Variablen

lokale Timer

Aktionsfolge

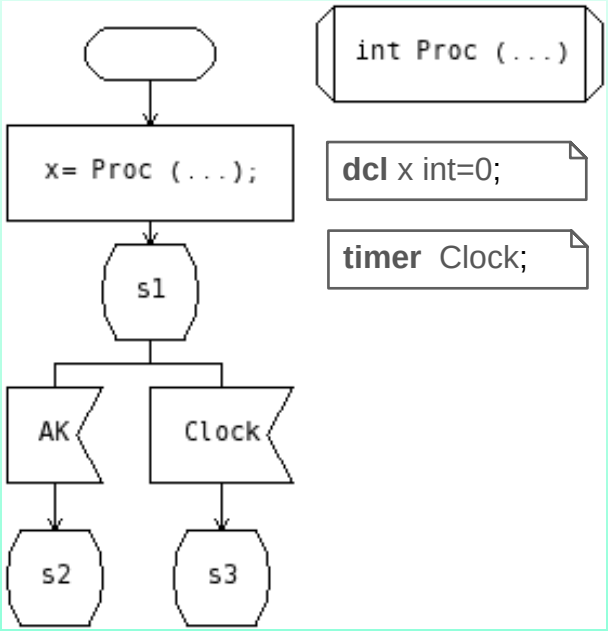
Zustandsüber-gangsgraph

Eingabe-Nachricht (Alternativen für aktuelle Nachricht im zugeordneten Eingabepuffer)

Äquivalente Syntaxformen

SDL/GR

process type PType



SDL/PR

```
process type Ptype;

        dcl x int= 0;
        timer Clock;

        procedure Proc ( … ) return int;
        …
        endprocedure Proc;

    start;
        task x= Proc (…);
        nextstate s1;

    state s1;
        input AK;
        nextstate s2;

        input Clock;
        nextstate s3;
     …
endprocess type Ptype;
```

SDL-96- Schlüsselwörter (1) 131 ohne ASN.1

- active
- adding
- all
- alternative
- and
- any
- as
- atleast
- axioms
- block
- call
- channel
- comment
- connnect
- connection
- constant
- constants
- create
- dcl
- decision
- default
- else
- endalternative
- endblock
- endchannel
- endconnection
- enddecision
- endgenerator
- endmacro
- endnewtype
- endoperator
- endpackage
- endprocedure
- endprocess
- endrefinement
- endselect
- endservice
- endstate
- endsubstructure
- endsyntype
- endsystem
- env

SDL-96- Schlüsselwörter (2)

- error
- priority
- fi
- from
- fpar
- generator
- imported
- input
- literal
- macrodefinition
- macroid
- map
- mod
- nameclass
- newtype
- nextstate
- procedure
- noequality
- none
- not
- now
- offspring
- operator
- operators
- or
- ordering
- out
- output
- package
- parent
- process
- provided
- redefined
- referenced
- refinement
- rem
- remote
- reset
- return
- returns
- revealed
- self
- sender

SDL-96- Schlüsselwörter (3)

- service
- set
- signal
- signallist
- signalroute
- signalset
- spelling
- start
- state
- stop
- struct
- substructure
- synonym
- syntype
- system
- task
- then
- this
- timer
- to
- type
- use
- via
- view
- viewed
- virtual
- with
- xor

nicht alle
davon
sind SDL/GR wirksam

(nur die blau markierten)

Vergleich der Automatenmodelle: UML-SDL

Gemeinsamkeiten

- Harels Konzept
- Beispiele dafür:
  - erweiterte endliche Zustandsautomaten (lokale Variablen, Timer, Nachrichtenpool, parametrisierte Nachrichten)
  - verschachtelte Zustände, History-Zustände
  - Vererbung von Automaten
  - Nachrichten-getriggerte Übergänge
  - Zustandsbedingte Übergänge
  - Übergangs-, Entry-, Exit- und Do-Aktionen

Unterschiede

- SDL besitzt präzise Semantik (als eine mögliche Form der Fixierung semantischer Variationspunkte von UML)
- Beispiele dafür:
  - Nachrichtenreihenfolge (Pufferverwaltung)
  - Automatenreferenzierung
  - Nachrichtenadressierung (Unicast, simuliertes Broadcast)
  - Nachrichtenübertragung, insbesondere Parameter-Übergabe
  - dynamische Generierung von Automaten
  - Exception-Handling
  - Verwaltung dynamischer Automatenensemble gleicher Bauart
  - Spontane Zustandsübergänge
  - Einschränkung zustandsbedingter Übergänge
- SDL erlaubt typ- und stellvertreter-basierte Instanziierung von Automaten

SDL ist ausführbar
UML ist nur im Prinzip ausführbar

Weitere fixierte semantische Variationspunkte

- Datentypen, Action-Syntax und Action-Semantik (C- Style),
  SDL erlaubt Werte- und Referenz-Semantik

- Beziehungen zwischen aktiven Klassen und Zustandsautomaten sind wohl-definiert
  → Agenten sind die Vereinigung von Aktiver Klasse und Zustandsautomat
  Mehrfachvererbung für Agenten ist ausgeschlossen

- Agenten besitzen systemweit 1-deutige Referenzen, deren Kenntnis initial nur lokal gegeben ist. *Referenzen werden zur Nachrichtenadressierung benutzt.*

- Keine Aussage zum *quantitativen* Zeitverbrauch der Zustandsübergänge.

- Unterscheidung zwischen Übertragungskanäle mit und ohne Zeitverbrauch.
  Nachrichtenkanäle arbeiten grundsätzlich fehlerfrei, *eine Überholung von Nachrichten ist ausgeschlossen*

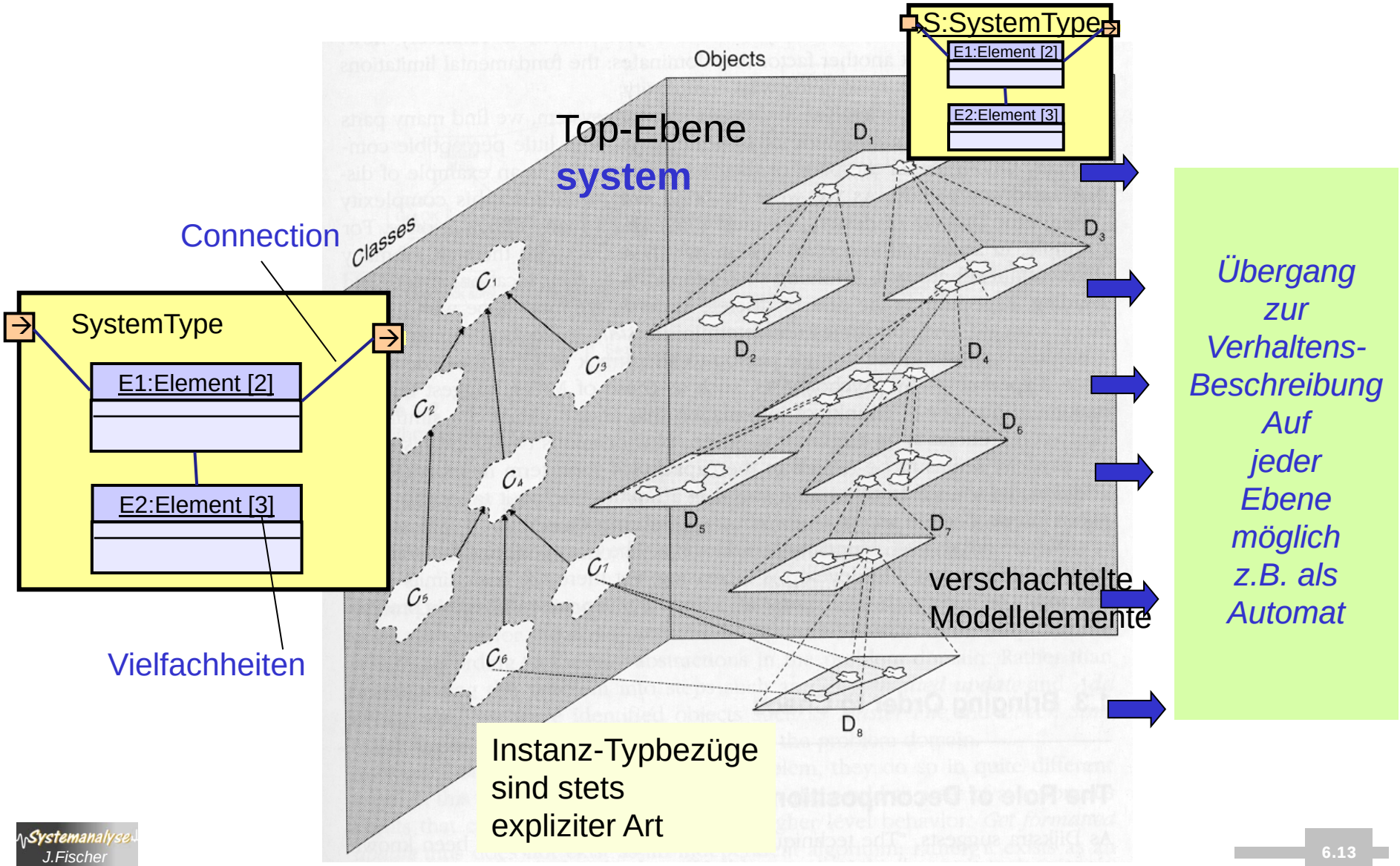
- Es gibt keine priorisierten Signale, dafür aber priorisierte Zustandsübergänge, *ist über Ersetzungsmodell präzisiert*

- Es gibt Remote-Procedure-Call, *ist über Ersetzungsmodell präzisiert*

- Systeminstanzen lassen sich definieren, und zwar *typbasiert* oder *stellvertreterbasiert*

Strukturelle Systemsicht in UML:

Klassen- und Objekthierarchien mit Wechselbeziehungen

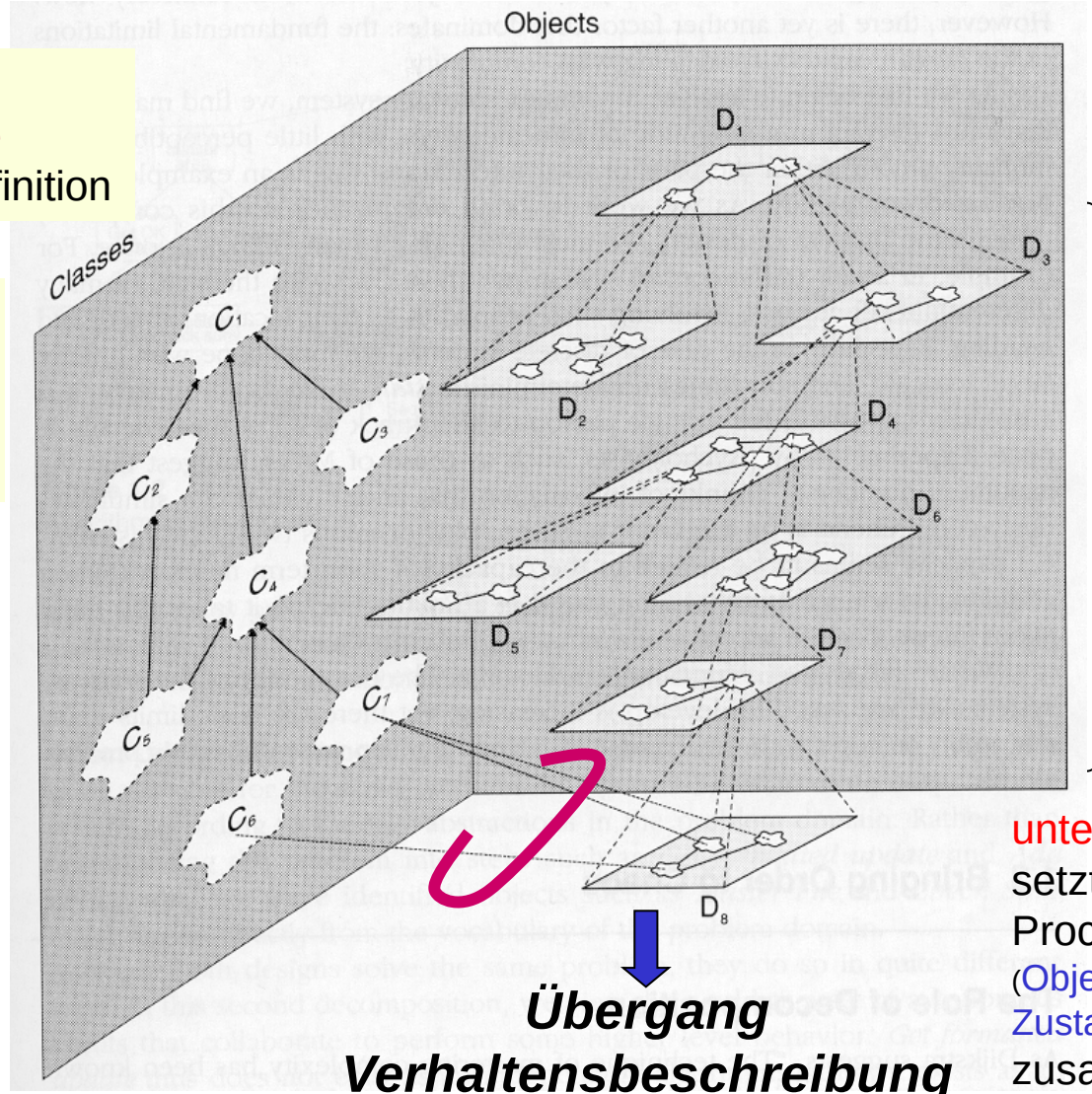


Strukturelle Systemsicht in SDL:

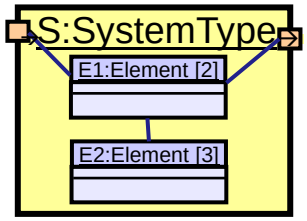
Klassen- und Objekthierarchien mit Wechselbeziehungen

Varianten:
- instanzbasierte
- typbasierte Definition

Instanz-Typbezüge können impliziter Art sein



oberster „Block“
system



verschachtelte Blöcke
block

unterster Block
setzt sich aus Process-Agenten
(Objekte aktiver Klassen + Zustandsmaschinen)
zusammen

Übergang Verhaltensbeschreibung