Modellbasierte Entwicklung von Technologien für selbstorganisierende dezentrale Informations-systeme im Katastrophen-management

METRIK

Projekt Erdbebenfrühwarnung
im SoSe 2011

HUMBOLDT-UNIVERSITÄT ZU BERLIN

Entwicklung verteilter echtzeitfähiger Sensorsysteme

Joachim Fischer
Klaus Ahrens
Ingmar Eveslage

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

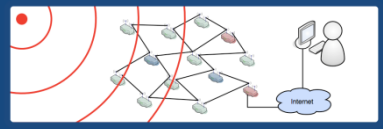
SAFER
Seismic eArly warning For EuRope

EDIM

SOSEWIN-extented

GFZ
Helmholtz-Zentrum
POTSDAM

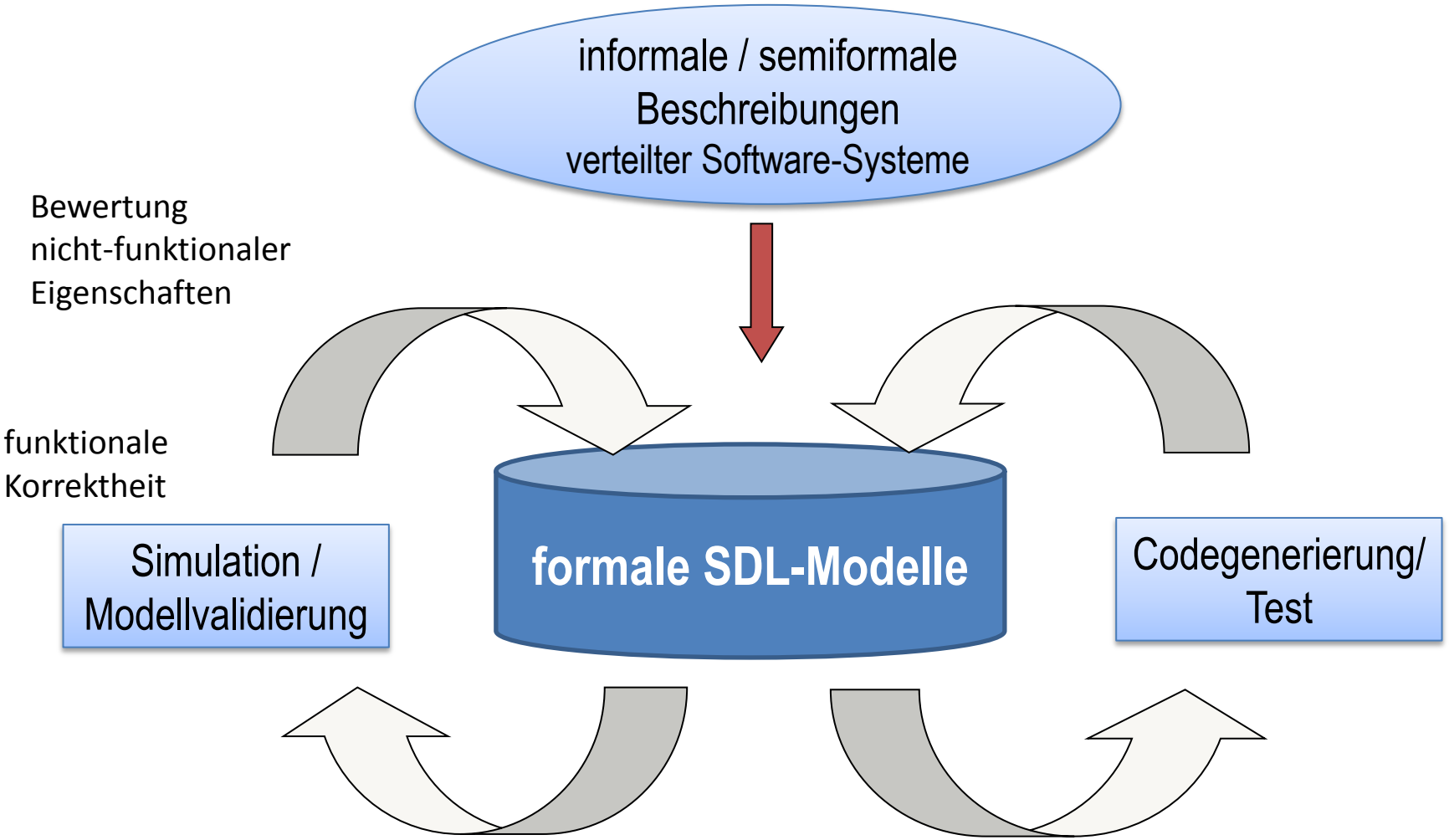
Systemanalyse



4. SDL als UML-Ersatz

1. UML und SDL-Zustandsmaschinen im Vergleich
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

RN-Protokolle → funktionale Korrektheit

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

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 (Harel), 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)

RN-Protokolle → funktionale + nichtfunktonale Korrektheit

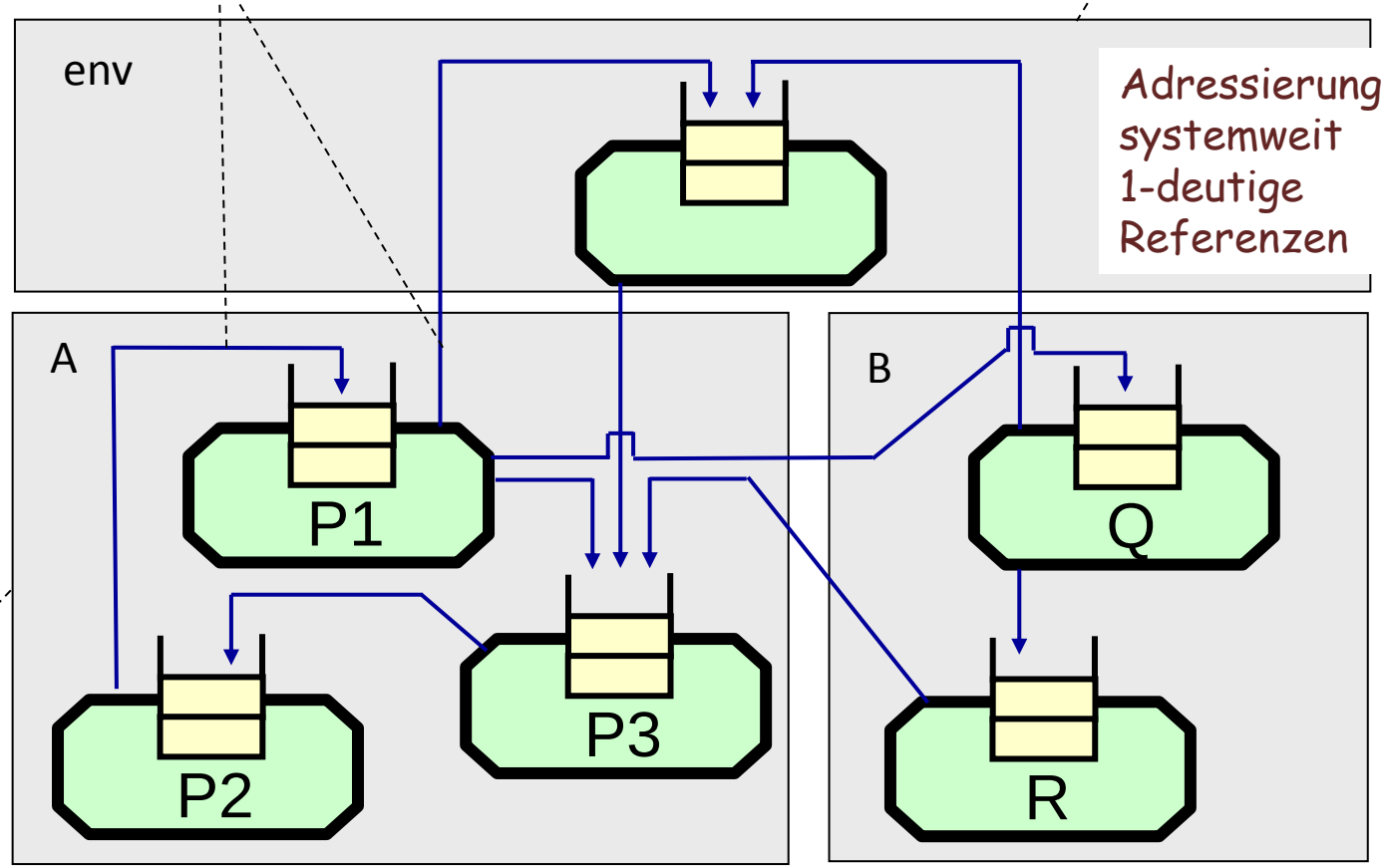
Laufzeitrepräsentation von SDL-Systemen

PType

potentieller Nachrichtenfluss über zeitlos bzw. zeitkonsumierende Übertragungskanäle

anonym strukturierte Systemumgebung

Agenten/ Prozesse können typbasiert oder repräsentanten-basiert definiert sein



Adressierung: systemweit 1-deutige Referenzen

Strukturen von Agenten/ Prozessen (hier: A und B

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

SDL-Basis

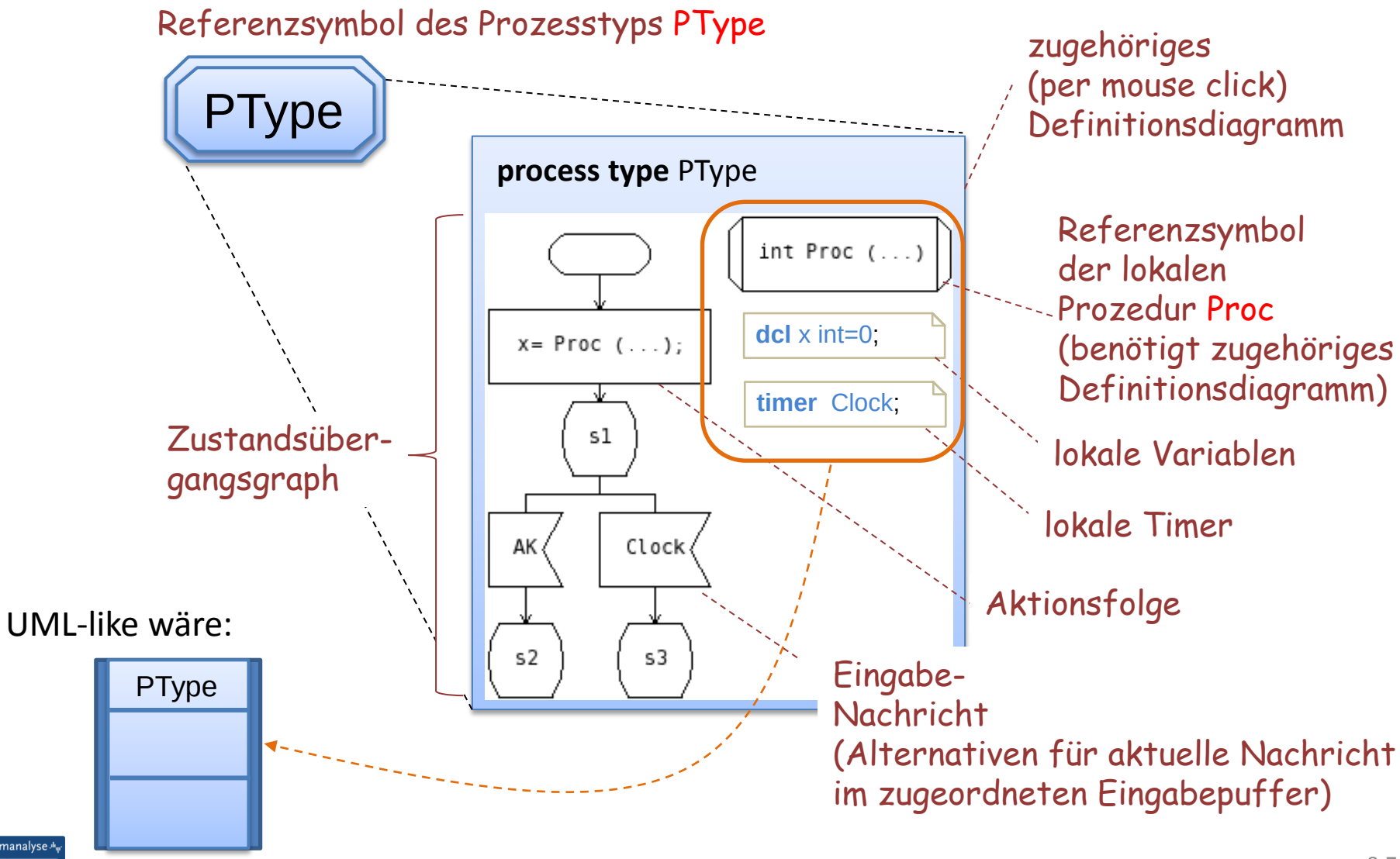
Agent= aktive Klasse mit Process-Beschreibung
- Classifier-Eigenschaft

- ein SDL-System besteht zur Laufzeit aus einer Menge von kommunizierenden Zustandsmaschinen
  definiert durch je einen Repräsentanten der festgelegten
  Process (Agenten)- Instanzmengen
  die in ihrer Wechselwirkung untereinander und mit der Umgebung des Systems das Verhalten erbringen

- die Wechselwirkungen werden über einen **asynchronen** Nachrichtenaustausch realisiert
  - Sender und Empfänger sind damit entkoppelt

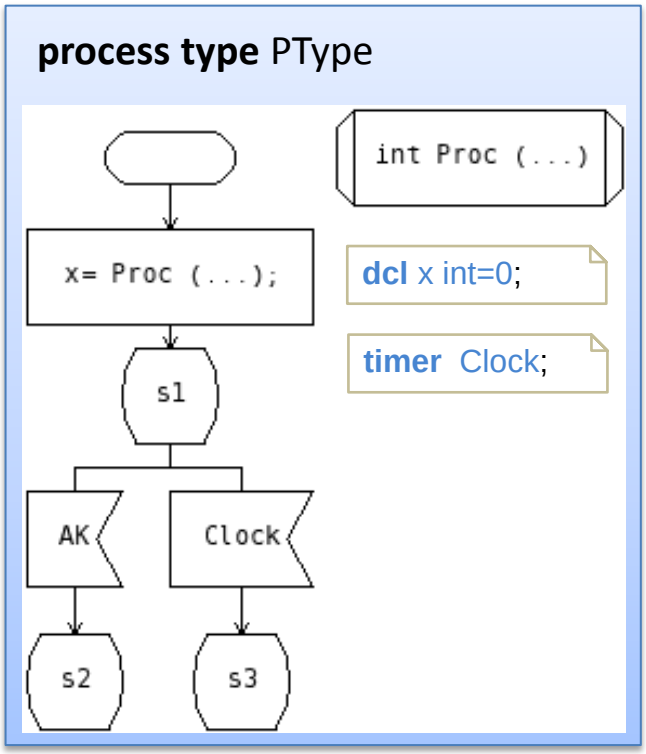
- jede Prozessinstanz besitzt (genau) einen Empfangspuffer zur Speicherung ankommender Nachrichten
  dieser ist idealerweise a priori unbeschränkt
  - keine Blockierung des Senders aufgrund eines vollen Puffers

Prozesse (Agenten): Typbeschreibung



Äquivalente Syntaxformen

SDL/GR GR=graphisch



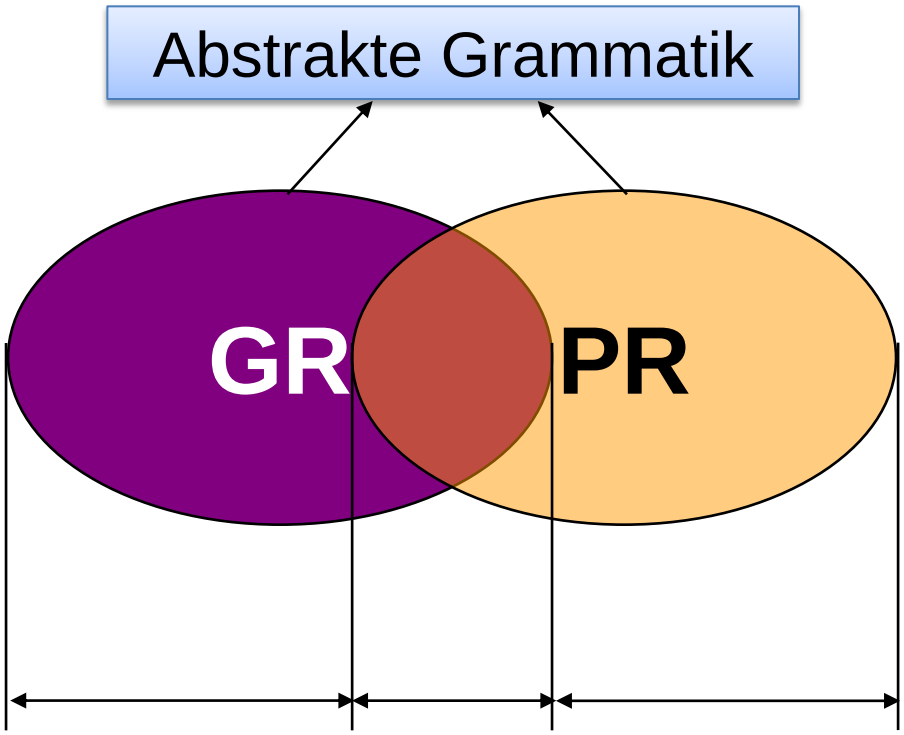

```
process type Ptype;

        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/PR

PR=programmiersprachlich

Repräsentationsformen

Z.100 (konkrete, abstrakte Syntax, statische u. dynamische Semantik)



CIF

Common Interchange Format (Z.107)

UML Profile

(Z.109)

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 Automatenkonzept
  - 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
- dynamische Generierung von Automaten
-

SDL ist ausführbar
UML ist nur im Prinzip ausführbar

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
  - Exception-Handling
  - Verwaltung dynamischer Automatenensemble gleicher Bauart
  - Spontane Zustandsübergänge
  - Einschränkung zustandsbedingter Übergänge
- SDL erlaubt typ- und stellvertreter-basierte Instanziierung von Automaten

Geklärte semantische Variationen in SDL

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

- Beziehungen zwischen aktiven Klassen und Zustandsautomaten
  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 Signaladressierung benutzt.*

- *Keine Aussage zum quantitativen Zeitverhalten der Zustandsübergänge.*
  Übertragungskanäle verbrauchen eine nicht spezifizierte Zeit

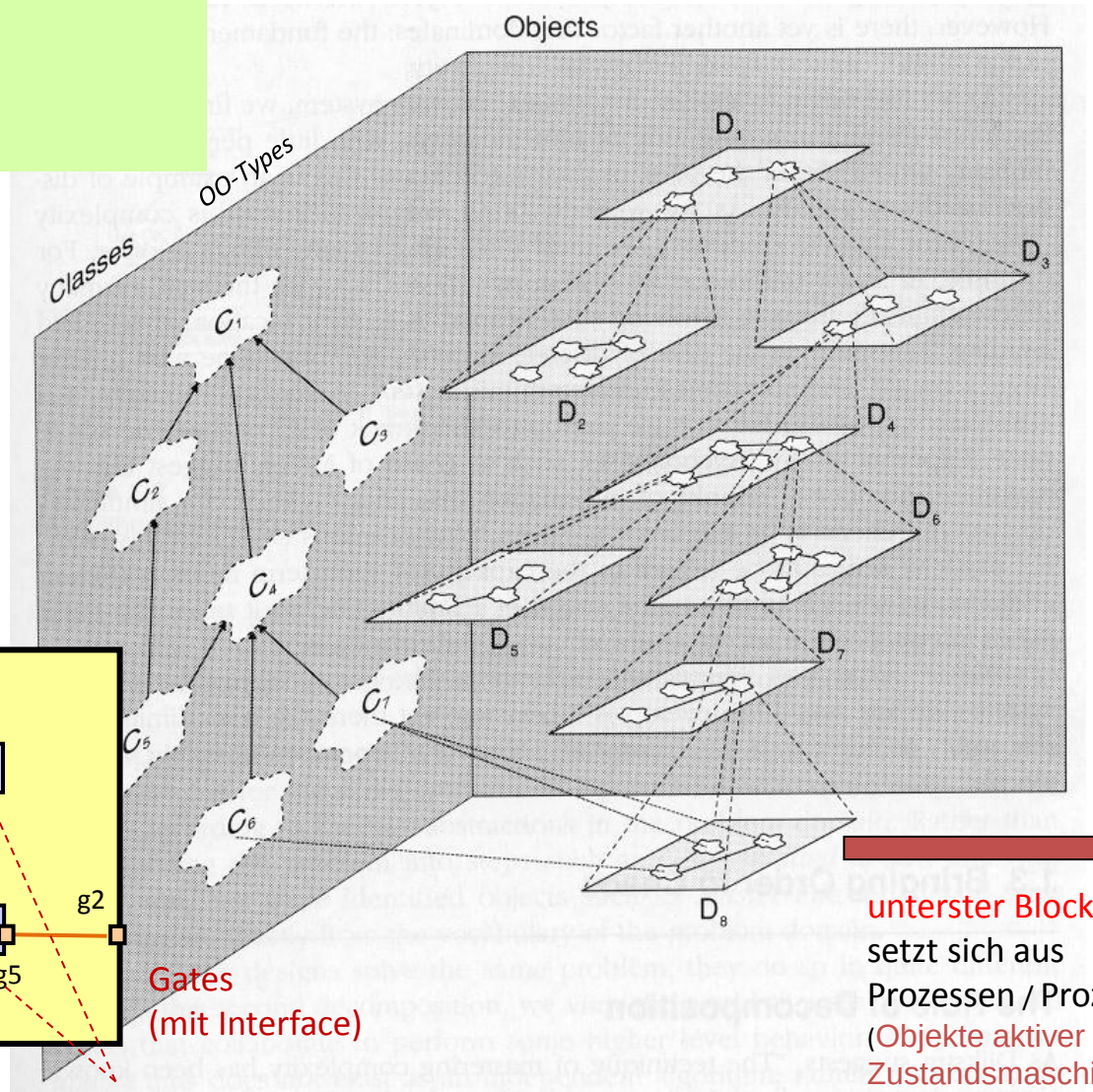
- Ereignisverwaltung (Signalempfangspuffer) ist präzisiert

- *Remote-Procedure-Call (guarded operation) ist über Ersetzungsmodell präzisiert*

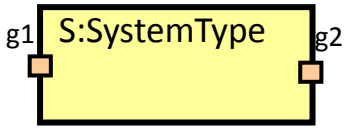
- *Systeminstanzen lassen sich definieren*

Die strukturelle Systemsicht in SDL

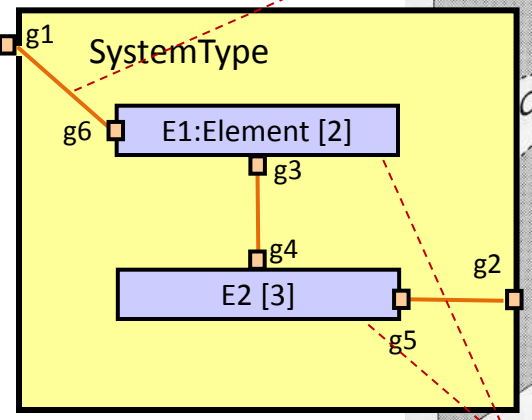
Instanz-Typbezüge können
- impliziter und
- expliziter Art
sein



oberster „Block“
System



Kanäle



Gates
(mit Interface)

Blockinstanzmengen

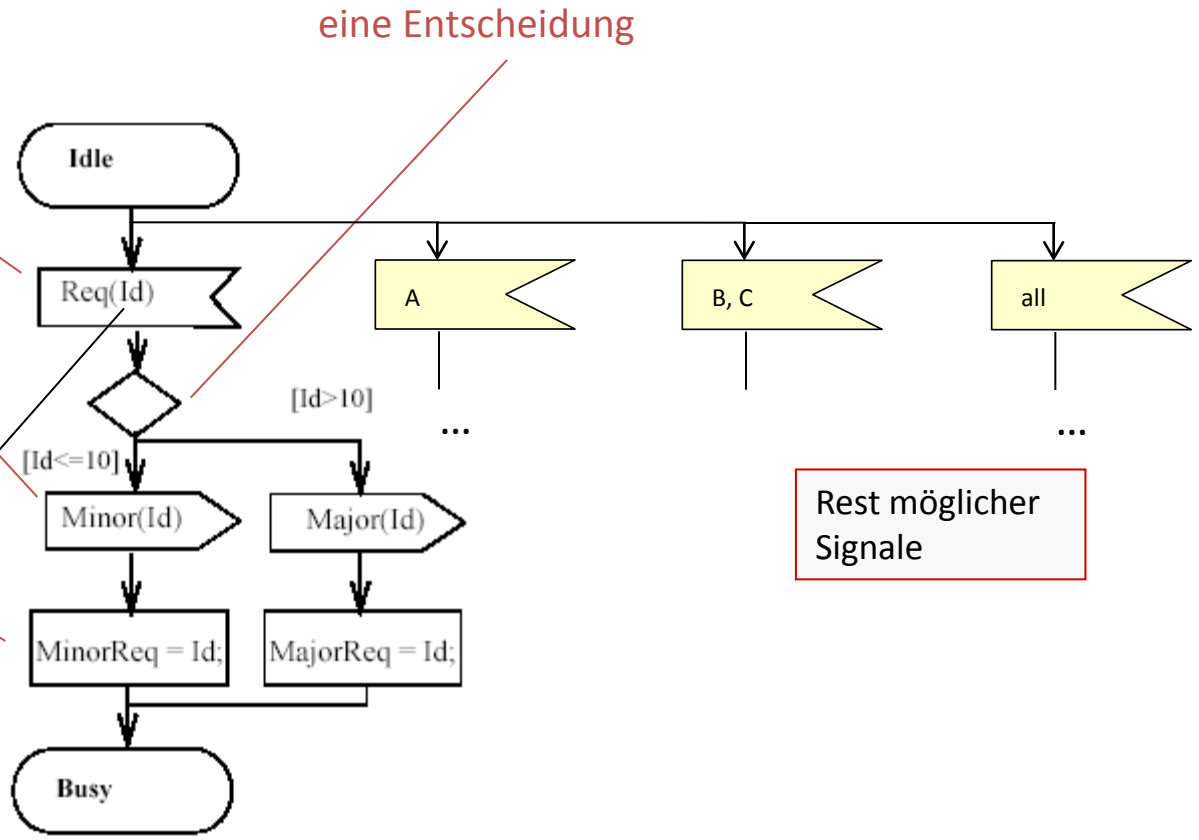
Übergang zur Verhaltens-Beschreibung

*nur auf **unterster** Ebene als Automat möglich*

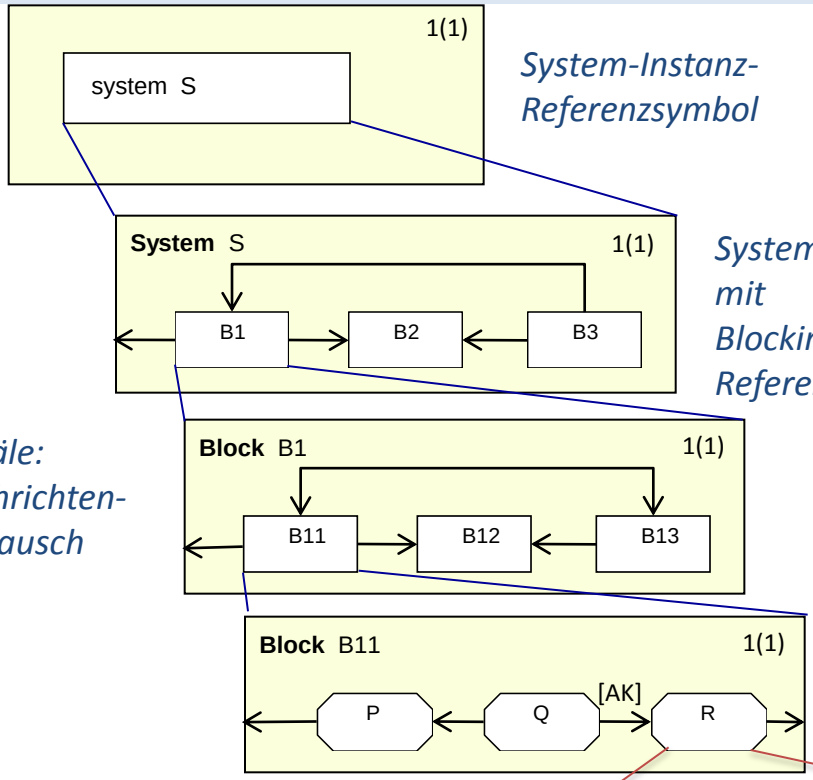
unterster Block
setzt sich aus
Prozessen / ProzessInstanzmengen
(Objekte aktiver Klassen + Zustandsmaschinen)
zusammen

UML erlaubt SDL-ähnliche Notation

- Empfangen eines Signals
- Senden eines Signals
- eine Aktionssequenz



SDL/GR - Syntax

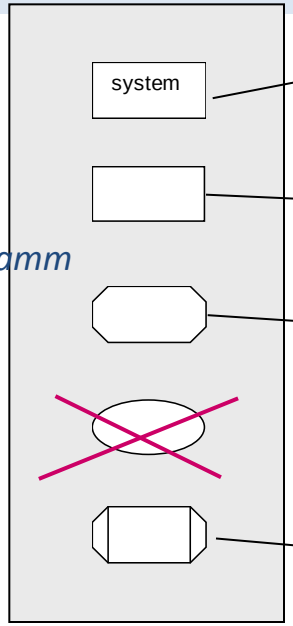


System-Instanz-Referenzsymbol

System-Diagramm mit Blockinstanz-Referenzen

Kanäle: Nachrichten-austausch

Referenzsymbole



System-Instanz

Block-Instanz(menge)

Prozess-Instanz(menge)
(Default-Kardinailität=1)

Service-Instanz

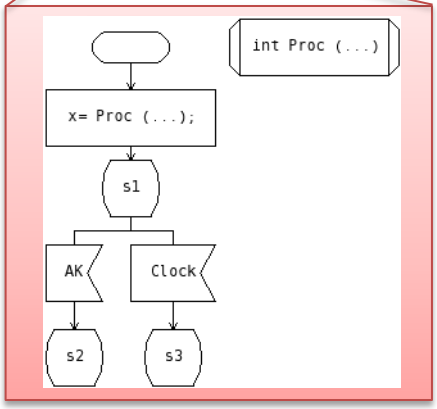
nicht in SDL-RT

Prozedur

SDL-Editor-Anforderung:
Diagramm- und Seitennavigation

Prozess als Zustandsautomat

- *kommunizieren per asynchronen Nachrichtenaustausch (impliziter Empfangspuffer)*
- *können andere Prozesse erzeugen*
- *Verhalten: endlich,unendlich*



- *lokale Variablen (auch Assoziationsenden)*
- *lokale Datentypen*
- *lokale Prozeduren/Funktionen*
- *lokale Zustände*
- *Zustandsübergänge als Ereignistrigger*

Systemsichtweisen von SDL

- **strukturelle Sichtweise**
  - Instanzsicht

    Beschreibung der Konfiguration eines Systems (system) bestehend aus funktionalen Einheiten (block) und ihren Relationen (channel) bei Identifikation der Systemgrenzen
  - Typsicht (falls für Instanzen/Instanzmengen Typen festgelegt sind)
  - Pakete (package) zur bequemen Wiederverwendung von Typen

- **verhaltensorientierte Sichtweise**
  - Verhalten einer funktionalen Einheit wird durch kommunizierende nebenläufig agierende Zustandsautomaten erbracht (agent/process)
  - Prozessverhalten: zeitdiskret
  - Strukturierungsmöglichkeiten von Verhaltensbeschreibungskonstrukten (procedure, service)