Modul OMSI-2
im SoSe 2010

Objektorientierte Simulation mit ODEMx

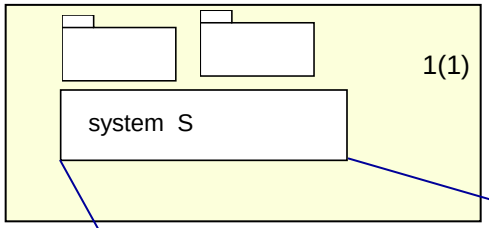
Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dipl.-Inf. Ingmar Eveslage
Dipl.-Inf. Andreas Blunk

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

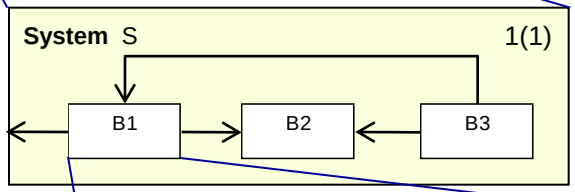
6. SDL

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/GR-Diagrammkomposition

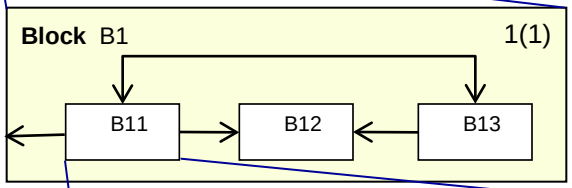


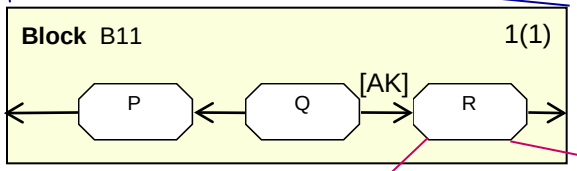
System-Instanz-Referenzsymbol



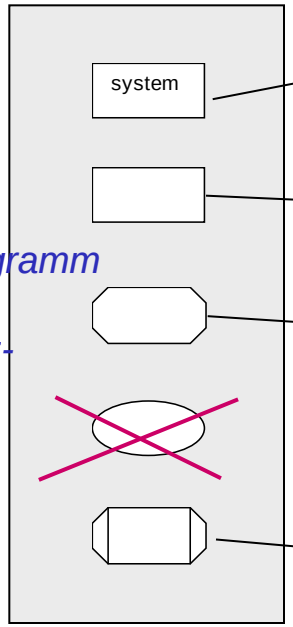
System-Diagramm mit Blockinstanz-Referenzen

Kanäle: Nachrichten-austausch





<u>benutzte Referenzsymbole</u>



System-Instanz

Block-Instanz(menge)

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

Service-Instanz

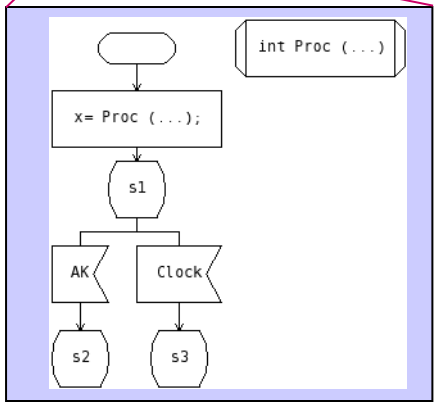
nicht mehr in SDL-2000

Prozedur

SDL-Editor-Anforderung:
Diagramm- und Seitennavigation

Prozess als Zustandsautomat

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



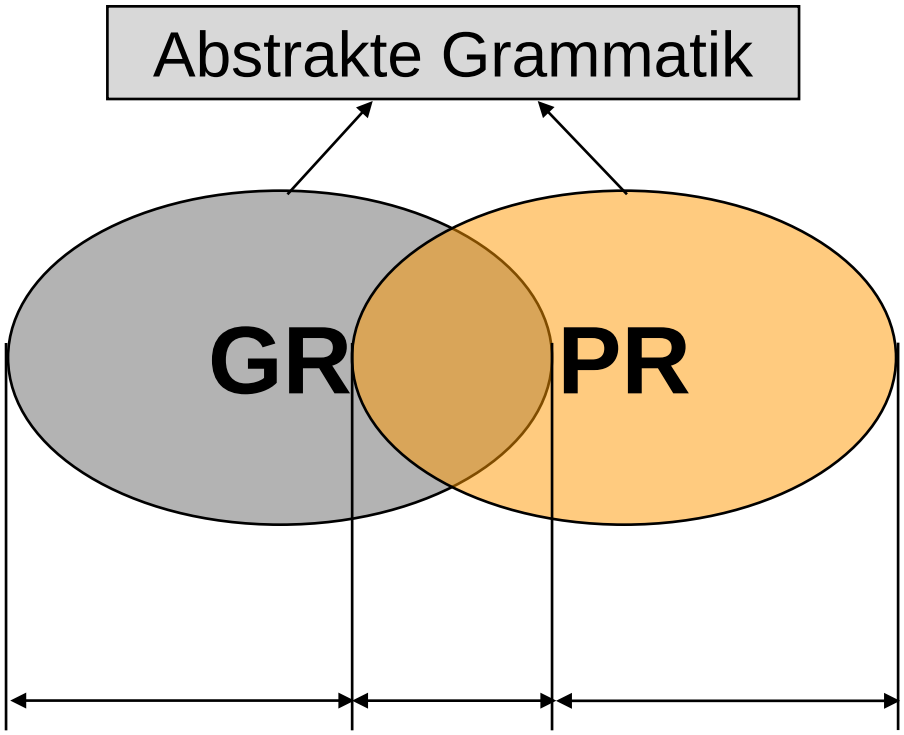
- *lokale Variablen (auch als Assoziation zu UML-Klassen)*
- *lokale Datentypen*
- *lokale Prozeduren/Funktionen*
- *lokale Zustände*
- *Zustandsübergänge per Nachrichtentrigge*

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-Repräsentationsformen

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



CIF

Common Interchange Format (Z.107)

UML Profile

(Z.109)

Der SDL-Sprachstandard (SDL-92/96)

Hauptdokument

Z.100
Sprachdefinition

Anhang A
Index

Anhang B
Konzepte

Datentypen

Anhang C
ACT ONE

Anhang D
package Predefined

Formale Semantik

SDL-92

Anhang F1
Notationen

Anhang F2
Statische Semantik

Anhang F3
Dynamische Semantik

SDL-96

Addendum 1
SDL-96

Systemsichtweisen

- **Strukturelle Sichtweise**
  - Instanzsicht

    Beschreibung der Konfiguration eines Systems (system) bestehend aus funktionalen Einheiten (block) und ihren Relationen (channel) bei Identifikation der Systemgrenzen
  - Typsicht (Klasse)
  - Pakete (package) zur bequemen Wiederverwendung von Typen (Klassen)

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

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

Überblick

SDL-96
(SDL-2000)
UML-2.0

- TAU-Werkzeug, IBM Rational
  (ursprünglich schwedische Firma TeleLogic)
- Cinderella (Standard-SDL, **SDL 96**, dänische Firma)
  mit Codegenerator der HU Berlin
- SDL-2000 ASM-Interpreter
  (MicroSoft, HU Berlin, Uni Kaiserslautern)
- PragmaDev Developer Studio (pragmatische SDL-Variante)
  - Eingeschränktes SDL
  - C/C++ Actionsprache
  - Kombination mit UML: KlassenDG, UseCaseDG, SequenceDG, DeploymentDG

  mit modifiziertem Code-Generator der HU Berlin
- SDL-Werkzeug auf UML-Basis nur rudimentär
  (HU Berlin)

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-Basis (bisheriges Fazit)

Agent = Instanz einer aktiven Klasse mit Process-Verhaltensbeschreibung
(In Form eines Erweiterten Endlichen Zustandautomaten)
- *mit UML-Classifier-Eigenschaft*

- ein SDL-System besteht zur Laufzeit aus einer Menge von kommunizierenden Zustandsmaschinen
  definiert durch je einen Repräsentanten der festgelegten
  Process (Agenten)- Instanzmengen

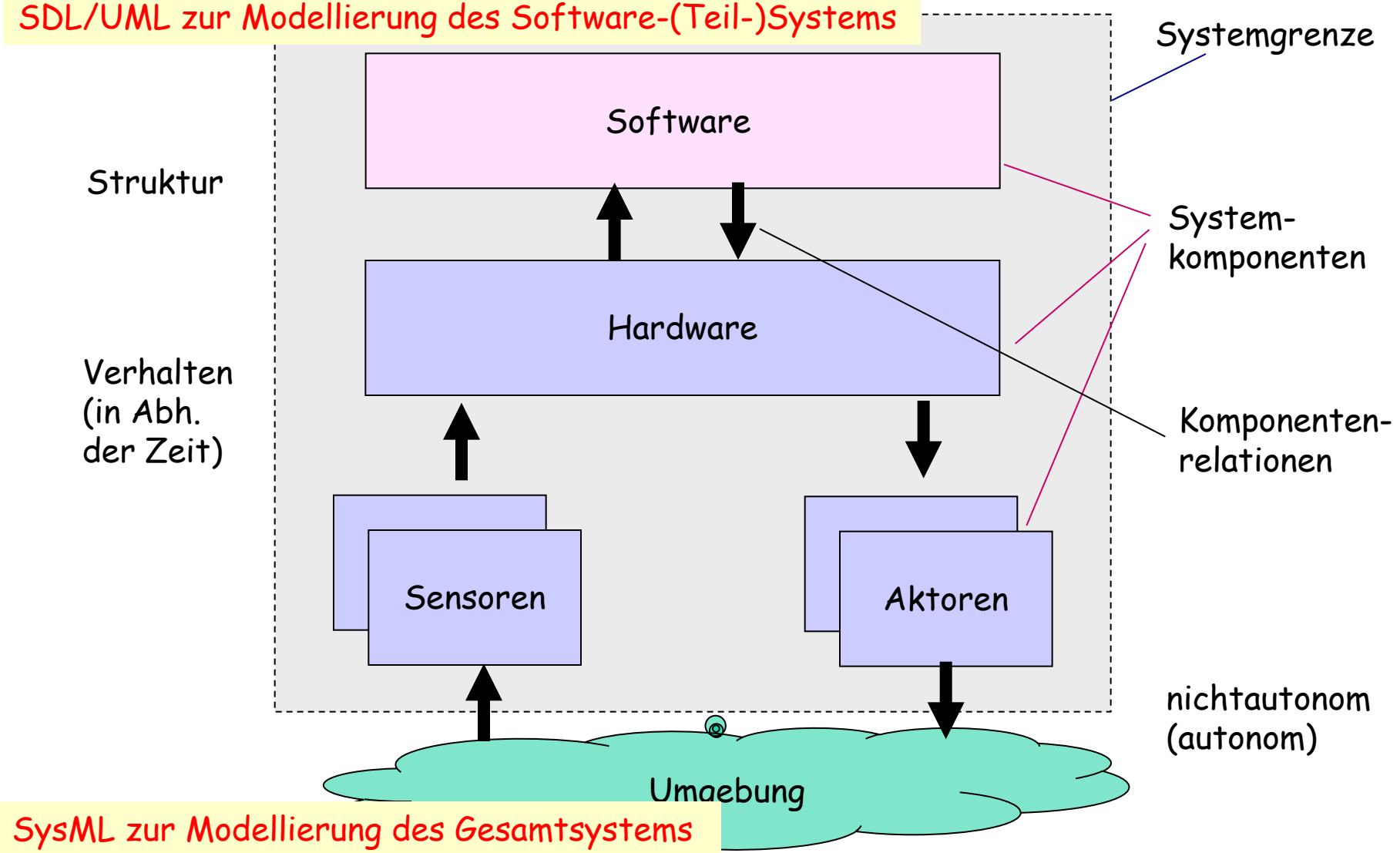
  die in ihrer Wechselwirkung untereinander und mit der
  Umgebung des Systems das Verhalten erbringen

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

- jede Prozessinstanz besitzt (genau) einen Empfangspuffer zur Speicherung ankommender Nachrichten
  dieser ist idealerweise a priori unbeschränkt
  - keine Blockierung oder ander Effekte des Senders aufgrund eines vollen Puffers (Abstraktion von der Realistät)

Eingebettete u. Reaktive Systeme und SDL

SDL/UML zur Modellierung des Software-(Teil-)Systems



SysML zur Modellierung des Gesamtsystems

Referenzierung und Lebenszeit von Agenten

- Ausgezeichneter Datentyp PId (*process identification*)
  - Werte des vordefinierten Datentyps PId stellen systemweit **eindeutige Referenzen** für Prozessinstanzen dar
  - Referenzen können zur **Nachrichtenadressierung** genutzt werden (alternative oder zusätzliche Angaben zu Kommunikationspfaden)

- Lebenslauf eines Prozesses
  - **Start** –(Pseudo-)Zustand, Einnahme (ohne zu verharren) per
    - Systemexistenz als initialer Prozess einer Prozessmenge
    - Erzeugung durch eine *process*-Instanz einer **anderen** Instanzmenge desselben Blockes oder der gleichen Instanzmenge während der Systemexistenzdauer
  - **Stop** , Einnahme (ohne zu verharren) führt zum Existenzende

Achtung: Laufzeitfehler, falls ein Nachricht an eine nicht mehr existente *process*-Instanz gesendet wird

- Systemexistenz
  - endet, sobald die letzte *process*-Instanz des Systems stoppt

Systemsichtweisen von SDL-96

stabile SDL-Version

- **Strukturelle Sichtweise**
  - Instanzsicht

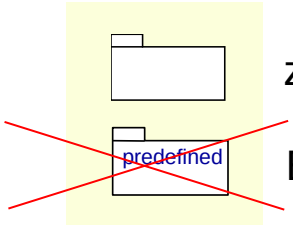
    Beschreibung der Konfiguration eines Systems (system) bestehend aus funktionalen Einheiten (block) und ihren Relationen (channel) bei Identifikation der Systemgrenzen

    Unterschiede: Block und Process
    (Verallgemeinerung als Agent in SDL-2000)
  - Typsicht (Klasse)
  - Pakete (package) zur bequemen Wiederverwendung von Typen (Klassen)
    vordefiniertes Paket für Datentypen

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

Weitere Referenzsymbole

- Paket (Package)

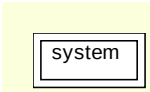


zur Zusammenfassung und Nutzung wiederverwendbarer Komponenten

Beispiel einer Paketreferenz: vordefinierte Datentypen

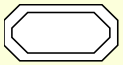
nicht in SDL-RT

- Strukturtypen (Vorlagen für Instanz- bzw. Instanzmengendefinition)

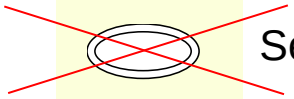


Systemtyp *nicht in SDL-RT*

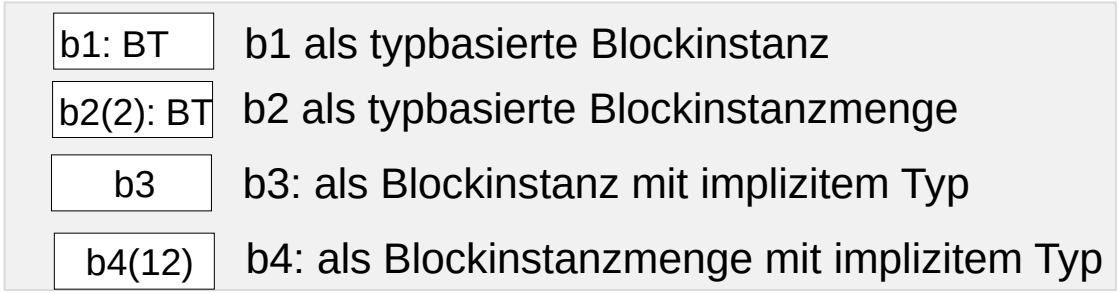
Blocktyp



Prozesstyp



Servicetyp *nicht in SDL-RT*



Inhalt folgt in SDL/RT der C-Syntax

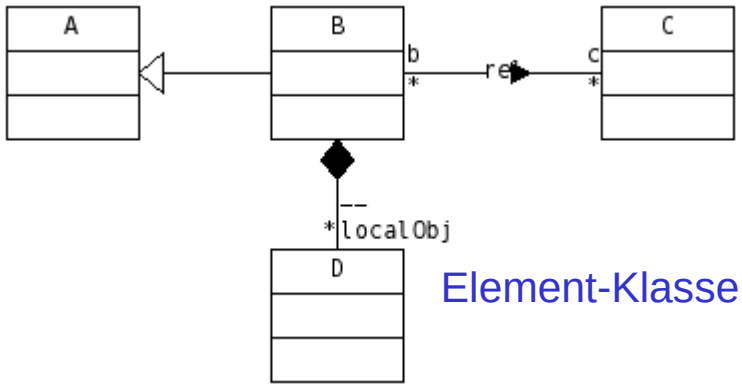
Achtung: Elementare Datentypen haben in SDL i.allg. **keine** Referenzsymbole werden als Text in universellen Textboxen erfasst

In SDL-RT haben jedoch passive (UML-)Klassen ein Referenzsymbol

Weitere SDL-Referenzsymbole

nur in SDL-RT

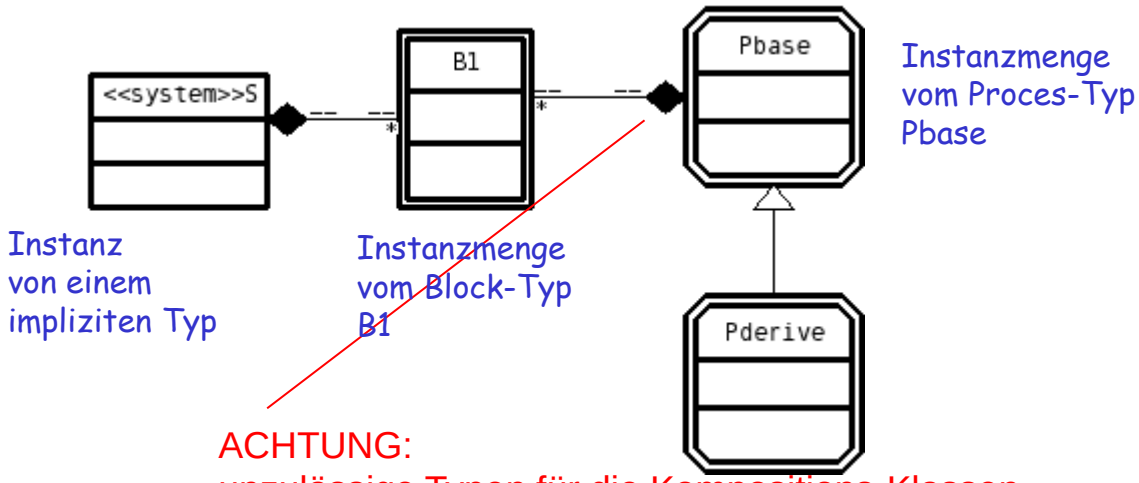
Container-Klasse



Klassendiagramm
(passive Klassen)

Element-Klasse, Lebenslauf der Elementobjekte ist an Existenz des Objektes der Container-Klasse gebunden

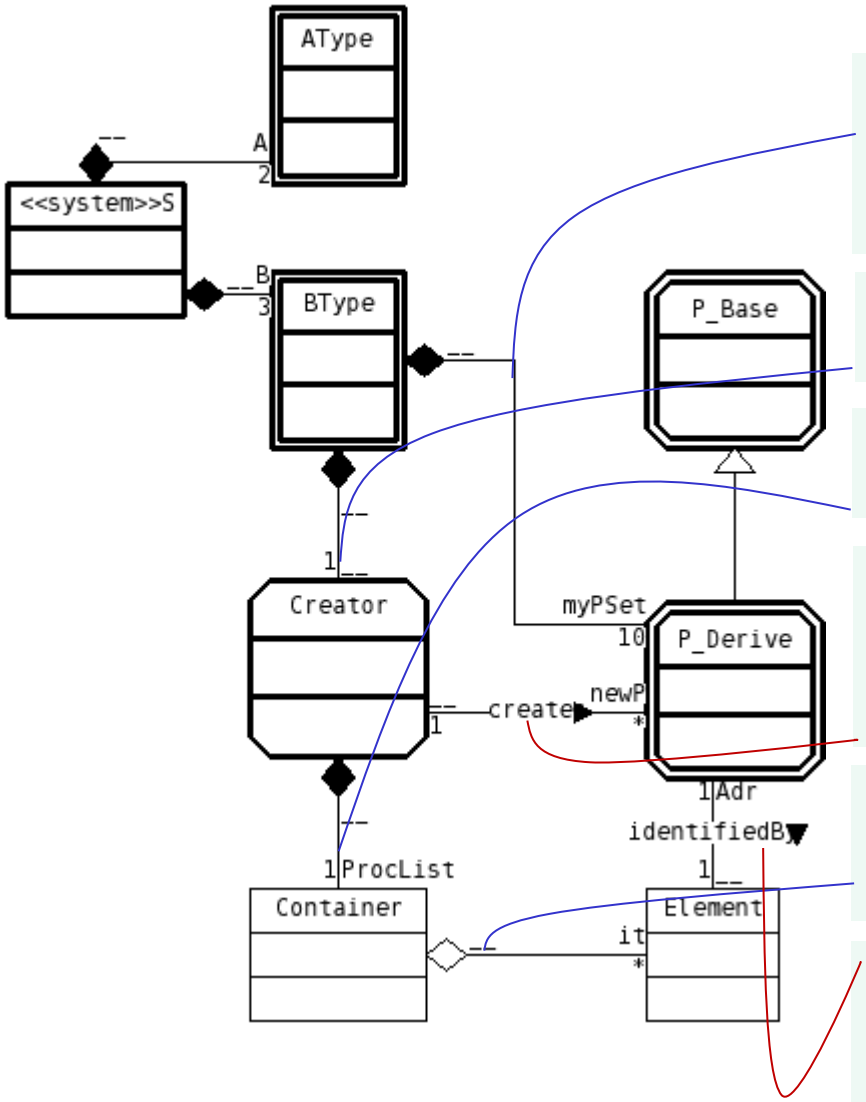
System Blocktyp Processtyp



Klassendiagramm
(aktive Klassen)

ACHTUNG:
unzulässige Typen für die Kompositions-Klassen
aus statischer SDL-Semantik

Anwendungsbeispiel



Komposition:
Instanz von P_Base (oder Ableitung)
gehört zu einer Instanz von BType
(zu deren lokalen Prozessmenge myPSet)

Komposition:
BType-Instanz besitzt Creator (als Process-Unikat)

Komposition:
Creator besitzt Container-Objekt ProcList (als Unikat)

Assoziation: creates
Creator erzeugt Instanzen von P_Derive
(beliebig viele, die in myPSet erfasst werden, deren Kardinalität auf 10 beschränkt ist)

Aggregation:
Container-Objekt verwaltet Element-Objekte (0,*)
erreicht die Elemente mit it (Iterator)

Assoziation: identifiedBy
über ProcList→it→Adr gelangt Creator an die Referenz (adr) aller P_Derive-Instanzen von ProcList
über newP gelangt Creator an die Referenz des zuletzt generierten Prozesses

Prozessidentifikation

in SDL-RT: RTDS_QueueId

- jede Prozessmengen-Instanz hat eine systemweit-eindeutige **Identifikation** vom Typ PId (PId = process identification)
- Vergabe eines PId-Wertes erfolgt implizit per Instanzgenerierung
- keine Kompatibilität von PId zu anderen Typen
- PId-Werte können nur
  - verglichen werden (==, =/) und
  - in Variablen vom Typ PId zur Adressierung von Nachrichten gespeichert werden
- einziges Literal (explizite Notation für einen Wert): null
- eine Prozessmengen-Instanz existiert mit Systemstart oder wird zur Laufzeit (durch einen anderen Prozess) explizit generiert
- die Lebensdauer einer Prozessinstanz bestimmt nur die Instanz selbst