

UML 2 Aktivitätsdiagramme

Syntax, Semantik, Pragmatik

Dr. Harald Störrle

MGM EDV-Beratung GmbH
& Uni München
stoerrle@informatik.uni-muenchen.de

©2005, Dr. Harald Störrle

Gliederung

- **Einleitung**
- **Syntax**
- **Semantik**
 - Probleme
- **Pragmatik**
- **Zusammenfassung**
- **Ausblick**

©2005, Dr. Harald Störrle

Gliederung

- **Einleitung**
- **Syntax**
- **Semantik**
 - Probleme
- **Pragmatik**
- **Zusammenfassung**
- **Ausblick**

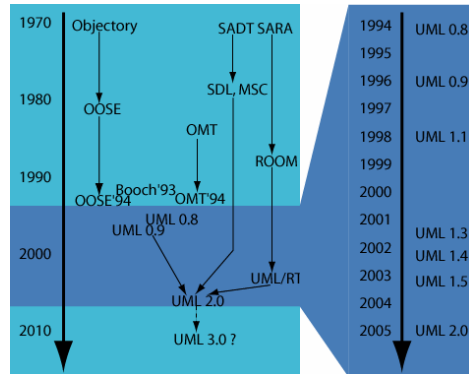
©2005, Dr. Harald Störrle

UML 2.0 – Bedeutung, Stand, Anspruch

- **Bedeutung**
 - „Lingua franca der Softwaretechnik“, Netzwerkeffekt
 - subsumiert, integriert und konsolidiert alle ihre Vorgänger
- **Stand der Standardisierung**
 - UML 2.0 seit März verabschiedet (Stand „2.10.2004“)
 - endgültige Version seit August verfügbar
 - Verabschiedung zog sich einige Jahre hin
 - komplett neues Metamodell
 - sauberer und vollständiger, viele Detailverbesserungen
- **Anspruch**
 - „Prototyp der nächsten Generation von Programmiersprachen“
Bran Selic: „The pragmatics of Model-Driven Development“
IEEE Software 20 (5) 2003

©2005, Dr. Harald Störrle

UML 2.0 – Vorgeschichte



©2005, Dr. Harald Störle

UML 2.0 – Diagrammtypen

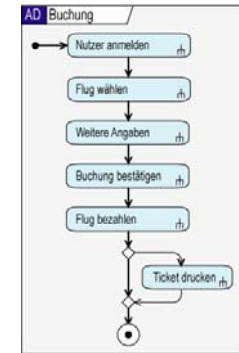
	Diagrammtyp	stellt schwerpunktmäßig dar	
Struktur	Klassendiagramm	statische Struktur (generisch / Momentaufnahme)	
	Montagediagramm	logische Systemstruktur	
	Komponentendiagramm	physische Systemstruktur	
	Verteilungsdiagramm	Rechnerinfrastruktur / Installation	
	Paketdiagramm	Aggregationshierarchie	
	Kollaboration	abstrakter struktureller Zusammenhang	
Verhalten	Nutzfalldiagramm	abstrakte Funktionalität	
	Aktivitätsdiagramm	Kontroll- und Datenfluss	
	Interaktion	Sequenzdiagramm	Nachrichtenaustausch über Zeit
		Kommunikationsdiagramm	Konfiguration interagierender Elemente
		Zeit(verlaufs)diagramm	koordinierter Zustandswechsel über Zeit
		Interaktionsübersicht	Fluss von Interaktionen
Zustandsautomat	ereignisgesteuerte Zustandsübergänge		

©2005, Dr. Harald Störle

Aktivitätsdiagramme – Anwendungsszenarien

• **Mit Aktivitätsdiagrammen können die verschiedensten Arten von Abläufen beschrieben werden:**

- Geschäftsprozesse
- Softwareprozesse
- Web-Services
- Workflows
- algorithmische Abläufe
- Programmabläufe
- ...

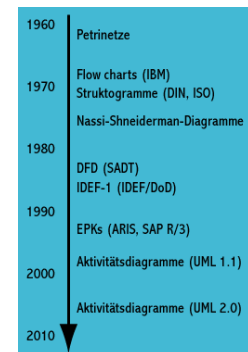


©2005, Dr. Harald Störle

Aktivitätsdiagramme – Vorläufer und Verwandte

• **Aktivitätsdiagramme haben viele Vorläufer/Verwandte:**

- Petrinetze
- Flow charts, Struktogramme
- Nassi-Shneiderman-Diagramme
- Programmablaufpläne (PAP)
- Datenflussdiagramme (DFDs)
- IDEF-1
- Ereignis-Prozess-Ketten (EPKs)
- UML 1.1 Aktivitätsdiagramme
- ...



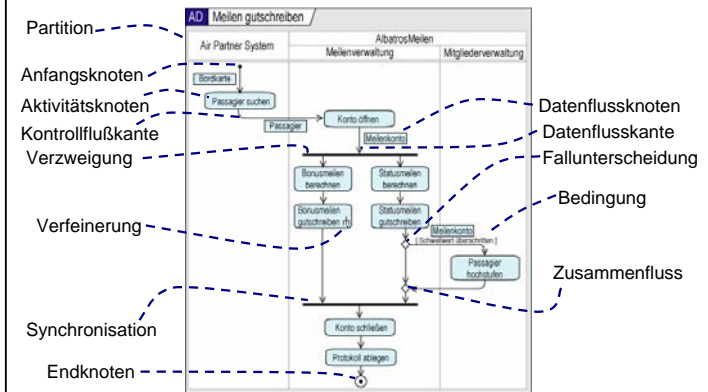
©2005, Dr. Harald Störle

Gliederung

- Einleitung
- Syntax
- Semantik
 - Probleme
- Pragmatik
- Zusammenfassung
- Ausblick

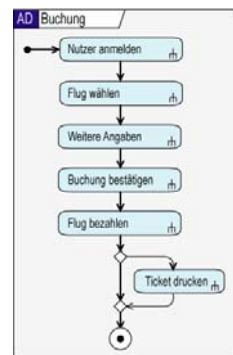
- einfacher Kontrollfluss
- einfacher Datenfluss
- Ausnahmen
- strukturierte Knoten
- Datenströme

Einfacher Daten- und Kontrollfluss

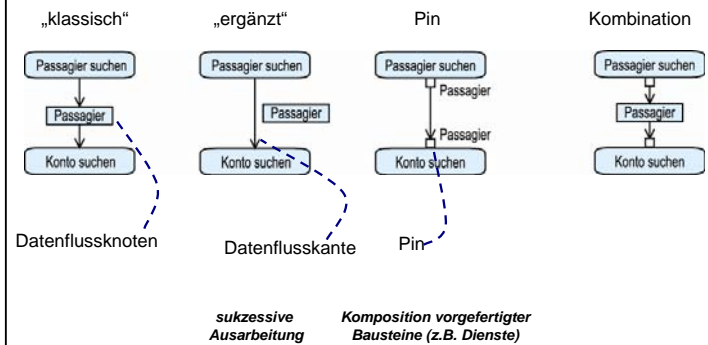


Übersicht

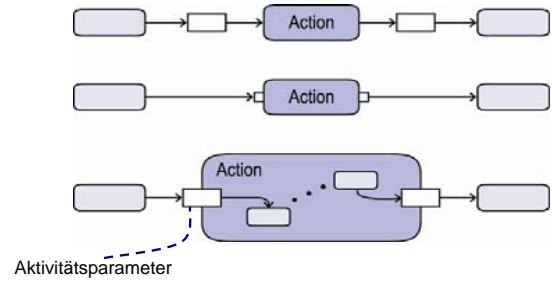
- Mit Aktivitätsdiagrammen lassen sich darstellen:
 - Kontrollfluss
 - Datenfluss
 - Makros
 - Auffaltung
- Neu in UML 2.0 sind:
 - Ströme
 - Massendaten („Collections“)
 - Strukturierte Knoten (Schleifen-, Bedingungs- & Auffaltungsknoten)
 - Ausnahmen



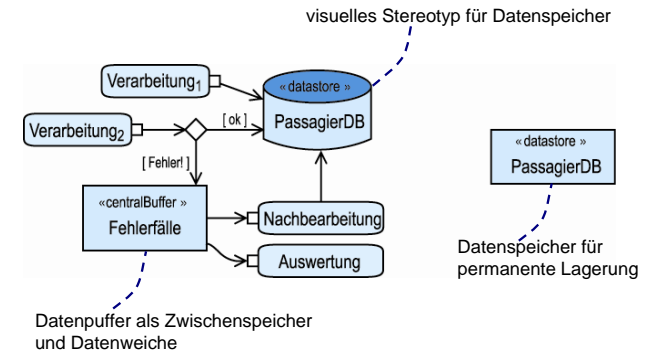
Notationsvarianten für einfachen Datenfluss



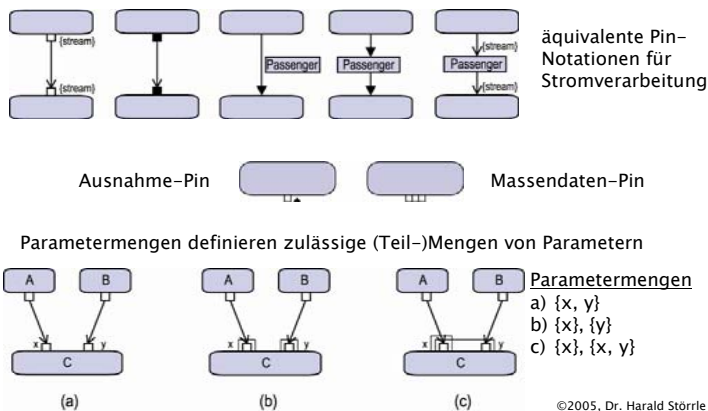
Pins als Parameter



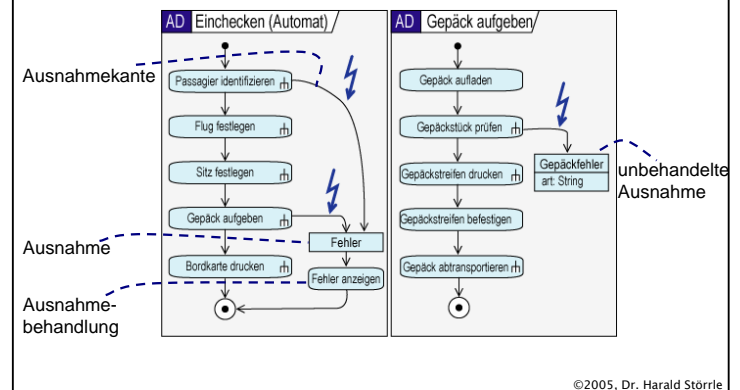
Arten von Datenspeichern



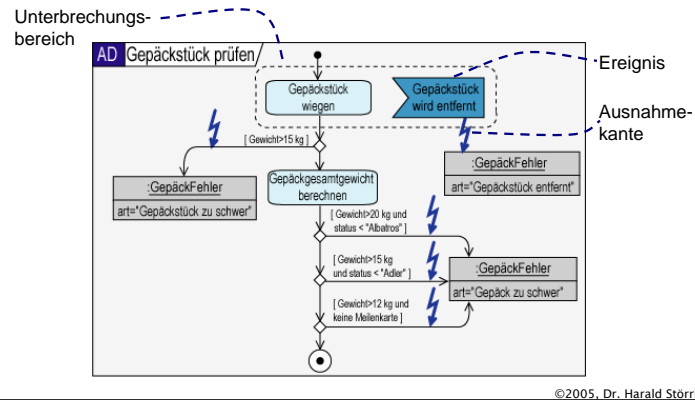
Arten von Pins



Ausnahme-Knoten

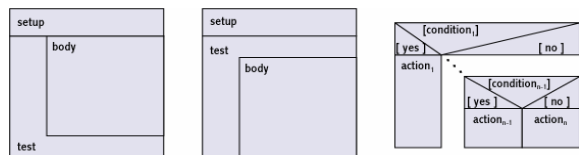


Ausnahme-Knoten



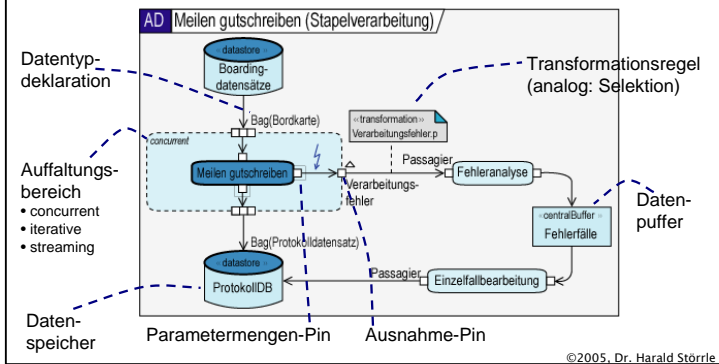
Strukturierte Knoten

- Strukturierte Knoten für Sequenz, Schleifen und Fallunterscheidung.
- Der UML-Standard definiert unzureichende Syntax (immer gleich, auch identisch zu Auffaltungsbereichen).
- Besser wäre z.B. Nassi-Shneiderman-artige Notation.



Auffaltungsbereiche

Mit Auffaltungsbereichen lässt sich Massendatenverarbeitung modellieren.



UML 2.0 – Vergleich mit anderen Notationen

	and/or/xor fork/join	non-wait nested Datenfluss	Prozeduraufruf	Conditional Nodes	Schleifen	Stromverarbeitung	Massendaten	Ausnahmen	Transaktionen
eEPK	✓	✗	(✓)	(✓)	✗	✗	✗	✗	✗
S/T-Netze	✓	✓	✗	✗	(✓)	✗	✗	✗	✗
UML 1.x ADs	✓	✗	✓	(✓)	✗	✗	✗	✗	✗
CP-Netze	✓	✓	✓	✗	(✓)	✓	✓	✗	✗
UML 2.0 ADs	✓	✓	✓	✓	✓	✓	✓	✓	✗
BP4WS	✓	✓	✓	✓	✓	✓	✓	✓	(✓)

©2005, Dr. Harald Störrie

Zwischenbilanz

- Die Syntax ist sehr ausdrucksmächtig, teilweise überladen.
- Obendrein gibt es oft viele verschiedene Wege, das Gleiche auszudrücken.
- Gleichzeitig gibt es zahlreiche neue bzw. nicht-traditionelle Konstrukte, die nur sehr mangelhaft definiert sind, semantisch wie syntaktisch.
- Eine formale Semantik kann unter anderem helfen, syntaktischen Zucker von essentiellen Ausdrucksmitteln zu unterscheiden (Beispiel strukturierte Knoten).

©2005, Dr. Harald Störrie

Motivation

- Semantik ist nie Selbstzweck.
- Eine Semantik zu definieren hat einen zweifachen Nutzen:
 - Die Formalisierung schärft die Definition und macht Fehler und Inkonsistenzen deutlich.
 - Die resultierende formale Semantik kann als Basis für Werkzeugunterstützung dienen.

©2005, Dr. Harald Störrie

Gliederung

- Einleitung
 - Syntax
 - Semantik
 - Probleme
 - Pragmatik
 - Zusammenfassung
 - Ausblick
- Ansatz
 - Petrinetze vs. CCS
 - (Formale Definition)
 - (Prozeduraufruf)
 - (Ausnahmen)

©2005, Dr. Harald Störrie

UML 2.0 – Dynamische Modelle

Diagrammtyp	Interaktions- diagramme	Statecharts	Aktivitäts- diagramme
Metaklasse	Interaction	StateMachine	Activity
Semantik laut Standard	„Traces of Event- Occurrences“	„Run-to- completion- steps“	„Petri-like“
Semantische Domäne	Formale Sprachen Σ^*	div.	Petrinetze <S, T, F>

©2005, Dr. Harald Störrie

Der Petrinetz-Ansatz

• **Klassisches Vorgehen**

– Abbildung von Elementen der abstrakten Syntax auf eine formale Domäne, hier von Aktivitäten nach Petrinetzen, wobei:

- **ausführbare Knoten** = Transitionen
- **Datenknoten** = Stellen
- **Kanten** = Flussrelation

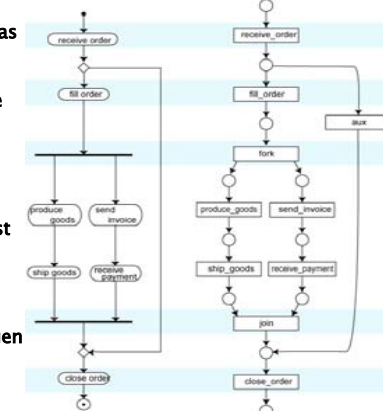
• **Vorteile**

- Abbildung ist nachvollziehbar und intuitiv
- Abbildung ist kompositional
- Ausgereifter Formalismus mit reicher Theorie
- zahlreiche Werkzeuge verfügbar
- Standard reklamiert „a Petri-like semantics“

Der Petrinetz-Ansatz – Vorteile

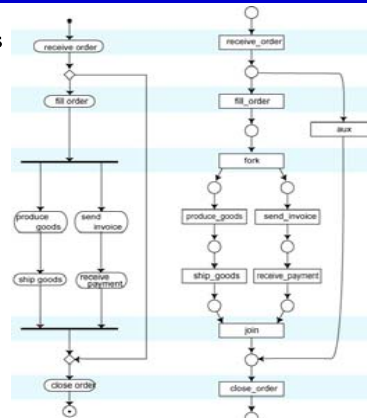
„Petri-like semantics“ – aber was heißt das?

- Die Semantik stellt eine direkte Beziehung zwischen Syntax (Aktivitätsdiagramm) und Semantik (Petrinetz) her.
- Dadurch ist die Semantik validierbar, und jedes Modell ist validierbar.
- Die Ausführung von Netz und Aktivitätsdiagramm können unmittelbar aufeinander bezogen werden („Traceability“).



Der Petrinetz-Ansatz

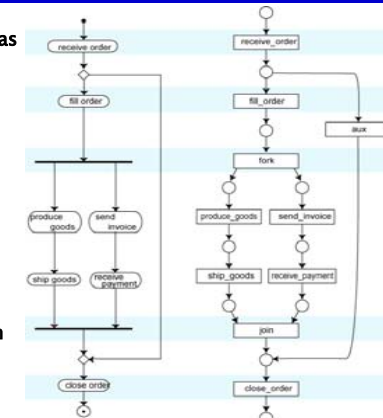
„Petri-like semantics“ – aber was heißt das?



Der Petrinetz-Ansatz – UML-Abdeckung

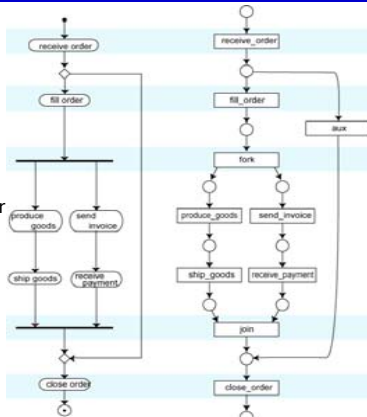
„Petri-like semantics“ – aber was heißt das?

- ✓ Einfache Deutung für Kontrollfluss,
- ✓ plausible Interpretation für Datenfluss,
- ✓ Prozeduraufruf, Ausnahmen, Schleifen, Bedingungsknoten lassen sich ebenfalls deuten,
- ✓ selbst Ströme und Massendaten lassen sich abbilden.

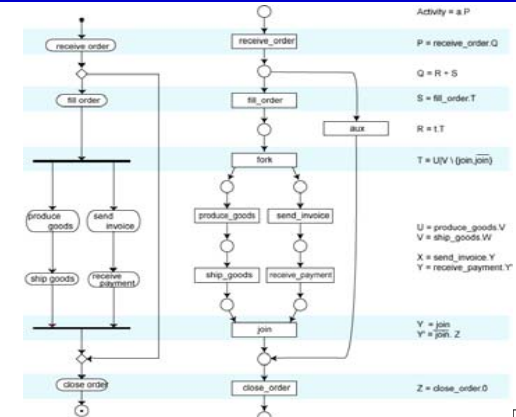


Eigenschaften – quantitativ

- Durch die Traceability können auch komplexere Eigenschaften problemlos von Petrinetzen auf Aktivitätsdiagramme hochgezogen werden.
- **Beispiel 1: Durchlaufzeit**
 - Für jede Action wird die Dauer als Ausführungszeit einer Transition modelliert.
- **Beispiel 2: Füllstand**
 - Für ein zeitbehaftetes PN kann die mittlere Füllung von Stellen berechnet werden.



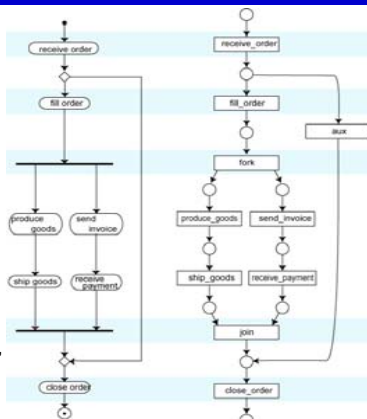
Der Petrinetz-Ansatz – vs. CCS



Dr. Harald Störrie

Eigenschaften – qualitativ

- Eigenschaften wie
 - Termination,
 - Erreichbarkeit,
 - Lebendigkeit u.v.m.
 sind für PN wohldefiniert, und können problemlos auf AD übertragen werden.
- Aufgrund der Struktur von Prozedurnetzen ist die Semantik außerdem kompositional, d.h. das klassische Petrinetz-Problem der Zustandsraum-Explosion wird wirkungsvoll eingedämmt, sowohl
 - für die Berechnung der Semantik,
 - als auch für die Analyse.

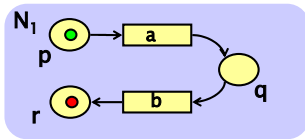


Petrinetze – Struktur & Verhalten

- Ein Petrinetz ist ein Tupel $\langle S, T, F, \bar{m}, \underline{m} \rangle$ mit
 - S Stellen $T \cap S = \emptyset$,
 - T Transitionen $F \subseteq T \times S \cup S \times T$,
 - F Flußrelation $\bar{m} \in S^*$ (Wort über S),
 - \bar{m} Anfangsmarkierung $\underline{m} \in S^*$.
 - \underline{m} Endmarkierung
- **Vorbereich (analog Nachbereich):**
 - ${}^{\circ}x = \{y \in S \cup T \mid \langle y, x \rangle \in F\}$ für $x \in S \cup T$.
- **Schaltregel**
 - Eine Transition t ist aktiviert in Markierung m („ $m \rightarrow^t$ “), wenn der Vorbereich belegt ist, also wenn $m \geq {}^{\circ}t$.
 - Wenn t in m aktiviert ist, kann t feuern und m' wird erreicht („ $m \rightarrow^t m'$ “), wobei $m' = m - {}^{\circ}t + t^{\circ}$.

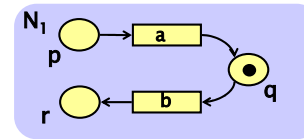
©2005, Dr. Harald Störrie

Petrinetze – Verhalten



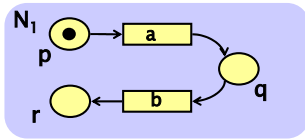
- Anfangsmarkierung
- Endmarkierung

Petrinetze – Verhalten



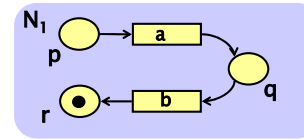
p
↓ a
 q

Petrinetze – Verhalten



p

Petrinetze – Verhalten



p
↓ a
 q
↓ b
 r

UML 2 Aktivitäten – Syntax, Semantik, Pragmatik Uni Paderborn, 14.9.2005 37

Verfeinerung von Aktivitäten

[[[]] : Activities → Prozedurnetze

konkrete Syntax
(Menge von Aktivitätsdiagrammen)

abstrakte Syntax
(Menge von Activities)

Semantik
(Menge von Petrinetzen mit Verfeinerungsfunktion: Netzsystem)

©2005, Dr. Harald Störle

UML 2 Aktivitäten – Syntax, Semantik, Pragmatik Uni Paderborn, 14.9.2005 39

Prozedurnetze – Verhalten

- **Call-Regel**
 - Sei t eine verfeinerte Transition (d.h. $\rho(t) = N$). Ein Schaltvorgang $call_{t,i}$ ist aktiviert in Markierung m („ $m \rightarrow^{call\ t\ i} m'$ “), wenn der Vorbereich von m belegt ist, also wenn $m \geq \rho t$, und i noch nicht verwendet wird.
 - Wenn $call_{t,i}$ in m aktiviert ist, kann $call_{t,i}$ stattfinden und m' wird erreicht („ $m \rightarrow^{call\ t\ i} m'$ “), wobei $m' = m - \langle c, j, m, f \rangle + \langle c, j, m - \rho t, f \cup \{i\} \rangle + \langle i, t, \bar{m}, \emptyset \rangle$.
- **Return-Regel**
 - Sei t eine verfeinerte Transition (d.h. $\rho(t) = N$). Ein Schaltvorgang $return_{t,i}$ ist aktiviert in Markierung m („ $m \rightarrow^{return\ t\ i} m'$ “), wenn die Verfeinerung die Endmarkierung erreicht hat und dort keine Aufrufe ausstehen, also wenn $\langle i, t, \bar{m}, \emptyset \rangle \in m$.
 - Wenn $return_{t,i}$ in m aktiviert ist, kann $return_{t,i}$ stattfinden und m' wird erreicht („ $m \rightarrow^{return\ t\ i} m'$ “), wobei $m' = m - \langle c, j, m, f \cup \{i\} \rangle + \langle c, j, m + \rho t, f \rangle - \langle i, t, \bar{m}, \emptyset \rangle$.
- **Anfangsmarkierung:** $\langle 1, \perp, \bar{m}, \emptyset \rangle$ für ein (ausgezeichnetes) N .

©2005, Dr. Harald Störle

UML 2 Aktivitäten – Syntax, Semantik, Pragmatik Uni Paderborn, 14.9.2005 38

Prozedurnetze – Struktur

Prozedurnetze [A. Kiehn, 1989]

- **Netzsystem:** $\langle N, \rho \rangle$
 - N Menge von Netzen
 - $\rho : (\bigcup_{N \in N} T_N) \rightarrow N$ Verfeinerungsfunktion
- **Zwei zusätzliche Feuerregeln:**
 - Feuern einer verfeinerten Transition instantiiert das Verfeinerungsnetz.
 - Erreichen der Endmarkierung eines Verfeinerungsnetzes löscht die Instanz.
- **Ein Zustand eines Netzsystems ist ein „Baum aus Markierungen“, formal: eine Menge aus Elementen $\langle id, caller, m, calls \rangle$ wobei**
 - **id** eindeutiger Identifier
 - **caller** Transition, die zu dieser Netzinstanz geführt hat (oder \perp)
 - **m** Markierung von ρ (**caller**)
 - **calls** Menge von Identifier von Netzinstanzen, die erzeugt wurden

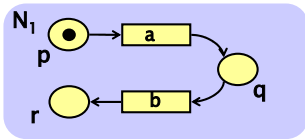
©2005, Dr. Harald Störle

UML 2 Aktivitäten – Syntax, Semantik, Pragmatik Uni Paderborn, 14.9.2005 40

Prozedurnetze – Verhalten

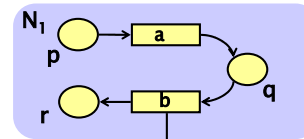
©2005, Dr. Harald Störle

Prozedurnetze – Verhalten



$\langle 1, \perp, p, \emptyset \rangle$

Prozedurnetze – Verhalten



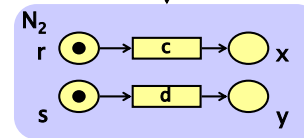
$\langle 1, \perp, p, \emptyset \rangle$

↓ a

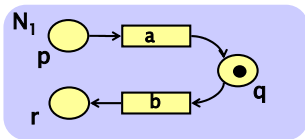
$\langle 1, \perp, q, \emptyset \rangle$

↓ call_b

$\langle 1, \perp, \varepsilon, \{2\} \rangle \langle 2, b, rs, \emptyset \rangle$



Prozedurnetze – Verhalten

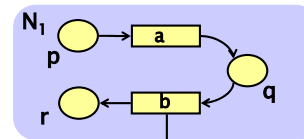


$\langle 1, \perp, p, \emptyset \rangle$

↓ a

$\langle 1, \perp, q, \emptyset \rangle$

Prozedurnetze – Verhalten



$\langle 1, \perp, p, \emptyset \rangle$

↓ a

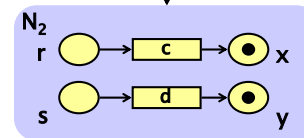
$\langle 1, \perp, q, \emptyset \rangle$

↓ call_b

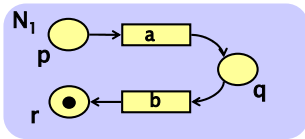
$\langle 1, \perp, \varepsilon, \{2\} \rangle \langle 2, b, rs, \emptyset \rangle$

↓ c, d

$\langle 1, \perp, \varepsilon, \{2\} \rangle \langle 2, b, xy, \emptyset \rangle$

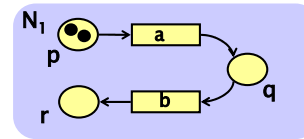


Prozedurnetze – Verhalten



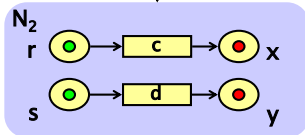
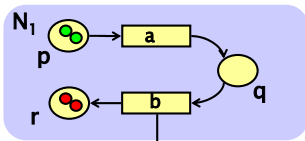
$\langle 1, \perp, p, \emptyset \rangle$
 $\downarrow a$
 $\langle 1, \perp, q, \emptyset \rangle$
 $\downarrow \text{call}_b$
 $\langle 1, \perp, \varepsilon, \{2\} \rangle \langle 2, b, rs, \emptyset \rangle$
 $\downarrow c, d$
 $\langle 1, \perp, \varepsilon, \{2\} \rangle \langle 2, b, xy, \emptyset \rangle$
 $\downarrow \text{return}_b$
 $\langle 1, \perp, r, \emptyset \rangle$

Prozedurnetze – Verhalten



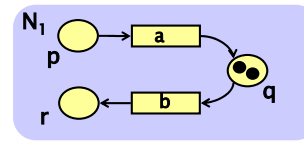
$\langle 1, \perp, pp, \emptyset \rangle$

Prozedurnetze – Verhalten



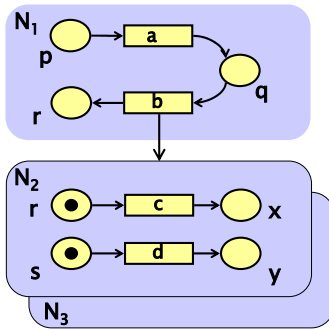
● Anfangsmarkierung
● Endmarkierung

Prozedurnetze – Verhalten



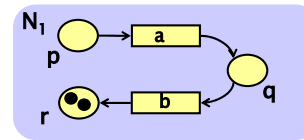
$\langle 1, \perp, pp, \emptyset \rangle$
 $\downarrow a, a$
 $\langle 1, \perp, qq, \emptyset \rangle$

Prozedurnetze – Verhalten



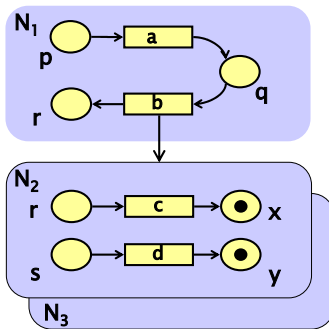
$\langle 1, \perp, pp, \emptyset \rangle$
 $\downarrow a, a$
 $\langle 1, \perp, qq, \emptyset \rangle$
 $\downarrow call_b, call_b$
 $\langle 1, \perp, \varepsilon, \{2,3\} \rangle$
 $\langle 2, b, rs, \emptyset \rangle \langle 3, b, rs, \emptyset \rangle$

Prozedurnetze – Verhalten



$\langle 1, \perp, pp, \emptyset \rangle$
 $\downarrow a, a$
 $\langle 1, \perp, qq, \emptyset \rangle$
 $\downarrow call_b, call_b$
 $\langle 1, \perp, e, \{2,3\} \rangle$
 $\langle 2, b, rs, \emptyset \rangle \langle 3, b, rs, \emptyset \rangle$
 $\downarrow c, c, d, d$
 $\langle 1, \perp, e, \{2,3\} \rangle$
 $\langle 2, b, xy, \emptyset \rangle \langle 3, b, xy, \emptyset \rangle$
 $\downarrow return_b, return_b$
 $\langle 1, \perp, rr, \emptyset \rangle$

Prozedurnetze – Verhalten



$\langle 1, \perp, pp, \emptyset \rangle$
 $\downarrow a, a$
 $\langle 1, \perp, qq, \emptyset \rangle$
 $\downarrow call_b, call_b$
 $\langle 1, \perp, \varepsilon, \{2,3\} \rangle$
 $\langle 2, b, rs, \emptyset \rangle \langle 3, b, rs, \emptyset \rangle$
 $\downarrow c, c, d, d$
 $\langle 1, \perp, \varepsilon, \{2,3\} \rangle$
 $\langle 2, b, xy, \emptyset \rangle \langle 3, b, xy, \emptyset \rangle$

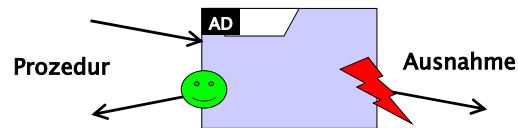
Wichtige Eigenschaften von Prozedurnetzen

- Prozedurnetze haben viele gute Eigenschaften:
 - Separat übersetzbar
 - Kompositional hinsichtlich Erreichbarkeit und Lebendigkeit
 - Werkzeugunterstützung ist als Erweiterung herkömmlicher Werkzeuge leicht darstellbar
 - Erweiterung auf höhere Petrinetze möglich (wenn auch technisch etwas mühselig)

Ausnahmen – Intuition

• **Grundidee**

- Es geht bei Ausnahmen um die Vermeidung von Gotos durch wohldefinierte Sprünge.
- Eine Ausnahme auszulösen ist praktisch identisch zum (vorzeitigen) Verlassen einer Prozedur.



©2005, Dr. Harald Störrle

Erweiterte Prozedurnetze

• **Einfache Erweiterung:**

- es gibt nicht nur eine reguläre Endmarkierung, bei deren Erreichen die Netzinstanz durch *return* gelöscht wird,
- es gibt auch noch eine Ausnahmemarkierung, bei deren Überdeckung die Netzinstanz durch *exit* gelöscht wird.

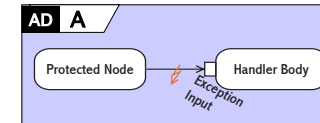
• **Exit-Regel**

- Sei t eine verfeinerte Transition (d.h. $\rho(t) = N$). Ein Schaltvorgang $exit_i$ ist aktiviert in Markierung m („ $m \rightarrow^{exit_i} m'$ “), wenn die Verfeinerung eine ausgezeichnete Stelle p_{exit} markiert hat, also wenn $\langle i, t, m' \rangle, calls \rangle \in m$ und $p_{exit} \in m'$.
- **Beachte:** $calls$ ist eine beliebige Menge von Identifiern!
- Wenn $exit_i$ in m aktiviert ist, kann $exit_i$ stattfinden und m' wird erreicht („ $m \rightarrow^{exit_i} m'$ “), wobei $m' = m - \langle j, c, m, f \cup \{i\} \rangle + \langle j, c, m + t^o, f \rangle - \langle i, t, m, sub \rangle - Calls(sub)$.

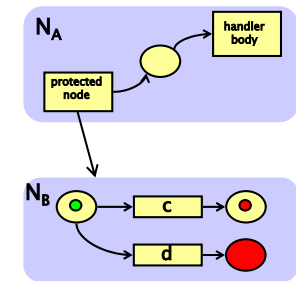
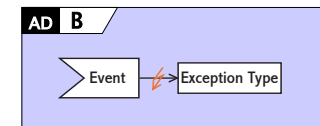
©2005, Dr. Harald Störrle

Semantik von Ausnahmeknoten

Ausnahmebehandlung



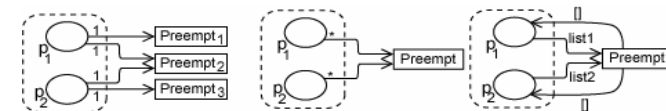
Ausnahmeauslösung



©2005, Dr. Harald Störrle

Semantik von Ausnahmeknoten

- **Ist denn ein neuer Petrinetz-Dialekt wirklich notwendig?**

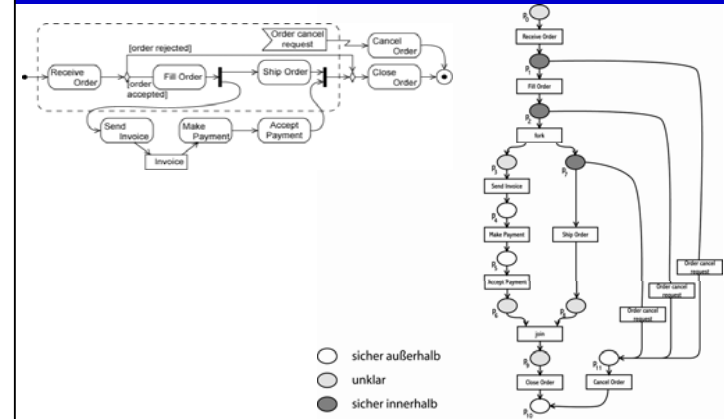


©2005, Dr. Harald Störrle

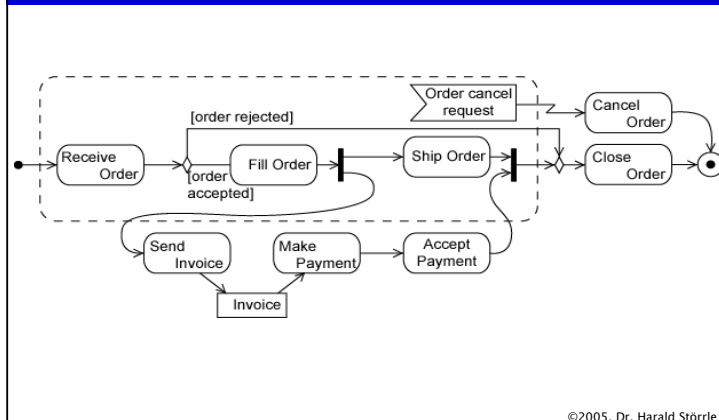
Gliederung

- Einleitung
 - Syntax
 - Semantik
 - Probleme
 - Pragmatik
 - Zusammenfassung
 - Ausblick
- unscharfe Abgrenzung
 - ungewollte Synchronisation
 - inkonsistente Zwischenzustände („traverse-to-completion“)
 - Interferenzen

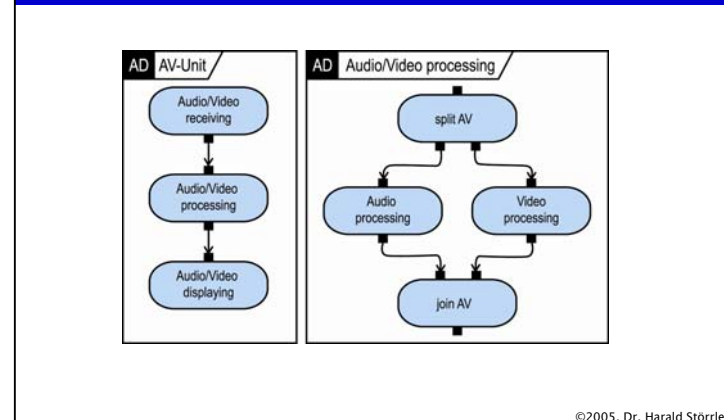
Problem 1: unscharfe Abgrenzung bei Ausnahmebehandlung



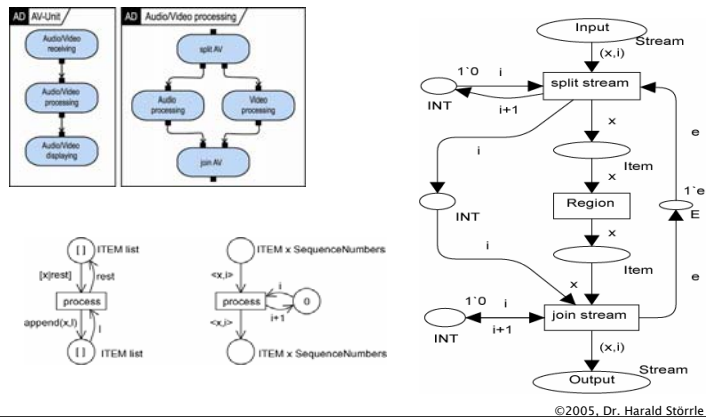
Problem 1: unscharfe Abgrenzung bei Ausnahmebehandlung



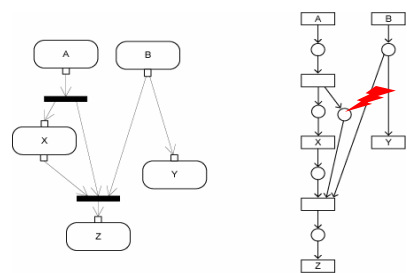
Problem 2: unerwünschte Synchronisation bei Datenströmen



Problem 2: unerwünschte Synchronisation bei Datenströmen



Problem 3: „Traverse-to-completion“



Im Petri netz ergeben sich Stellen, die zu Kontrollknoten korrespondieren, die UML verbietet aber Pufferung in Kontrollknoten.

Problem 4: Interferenzen

- Die einzelnen Ausdrucksmittel sind bislang jeweils nur für sich untersucht worden, jeweils mit einem geeigneten Petri netz-Dialekt:
 - Kontrollfluss: S/T-Netze
 - Prozeduraufruf: Prozedurnetze
 - Datenfluss: gefärbte Netze
 - Auffaltungen: gefärbte Netze
- Es gibt keinen Petri netz-Dialekt, der alles abdeckt, geschweige denn entsprechende Algorithmen oder Werkzeuge.

Problemanalyse

- Die tiefere Ursachen dieser Probleme liegt darin, dass eine operationale Interpretation von Aktivitätsdiagrammen vorgenommen wurde.
- Diese Interpretation führt Artefakte ein.
- Jeder operationale Ansatz hat dieses Problem (unterschiedlich stark).

Alternativen

- **Definition von Algorithmen direkt auf dem Modell**
 - sehr beschränkt, aber pragmatisch
- **Abbildung auf andere konventionelle semantische Domäne**
 - verlieren schöne Eigenschaften des Petrinetz-Ansatzes
 - z.B. Prozessalgebra: *graphischer Appeal geht verloren*
 - z.B. Partielle Worte o.ä.: *sehr großer Abstand zu Aktivitätsdiagrammen*
 - z.B. Transitionsysteme: *Kompositionalität, Abstraktion, Nebenläufigkeit ... ohne die Probleme zu lösen.*
- **Abbildung auf „esoterische“ semantische Domänen**
 - keine Theorie, keine Werkzeuge, keine Literatur, ...
- **Aktivitätsdiagramm als lose Spezifikation**
 - logische Charakterisierung (siehe Bock & Gruninger)
 - steckt noch in den Anfängen
 - verliert den graphischen Appeal
 - für viele praktische Zwecke ungeeignet
 - Abwicklung von Aktivitätsdiagrammen analog zu Petrinetz-Prozessen

©2005, Dr. Harald Störrie

Zwischenbilanz

- **Erkenntnis:**
 - UML Aktivitäten und Petrinetze passen nicht unmittelbar und vollständig zusammen.
- **Mögliche Schlußfolgerungen:**
 1. Ein anderer Formalismus muss gefunden werden, der besser zu UML paßt.
 2. Die UML muss sich ändern, so dass sie zu gängigen Formalismen (z. B. Petrinetzen) paßt.
 3. Es müssen verschiedene Semantiken je nach Verwendungszweck bzw. Absicht definiert werden.

©2005, Dr. Harald Störrie

Zwischenbilanz

- **Die vorgestellte Petrinetz-Semantik**
 - ist intuitiv und deckt UML 2 ADs nahezu vollständig ab,
 - die Abbildung ist nachvollziehbar und kompositional, und
 - der Ansatz kann heute praktisch verwendet werden.
- **Aber**
 - es gibt erhebliche Abweichungen vom Standard in wichtigen semantischen Details (teilweise Interpretationssache), und
 - sie ist nur abschnittsweise definiert, die Kombination verschiedener Semantik-Teilbereiche führt zu einem aufgeblähtem Formalismus.
- **Alternative Ansätze**
 - haben andere Probleme
 - sind bei weitem noch nicht so weit wie dieser (operationale) Ansatz.

©2005, Dr. Harald Störrie

Der nächste Schritt

- **Motivation**
 - Die Bedeutung einer Sprache entsteht aus ihrer Verwendung. Daher gibt es keine allgemeingültige Semantik für (alle) Aktivitätsdiagramme: für jeden Anwendungsbereich gibt es einen eigenen Formalismus (eine eigene Zielsprache) mit einer eigenen Semantik.
 - **Vorteil**
 - ADs können relativ zur Zielsprache sinngemäß anders interpretiert werden.
 - **Nachteil**
 - Wir müssen mit einer lieb gewordenen Vision brechen.
 - Wir müssen die Anwendungsbereiche finden und festschreiben.
- ➔ **UML-Semantik ist komplizierter, als wir bislang angenommen haben.**

©2005, Dr. Harald Störrie

Gliederung

- Einleitung
- Syntax
- Semantik
 - Probleme
- Pragmatik
- Zusammenfassung
- Ausblick

- Prozesse
- Algorithmen
- Schaltungen

©2005, Dr. Harald Störrie

Anwendungsbereiche für Aktivitätsdiagramme

- **Aktivitätsdiagramme könnten verwendet werden zur Beschreibung von jeder Art von Abläufen**
 - (betriebliche) Arbeitsabläufe
 - Software- und Geschäftsprozesse (z.B. EPKs, DFDs)
 - algorithmische Abläufe
 - Batchläufe (z.B. PHP, Perl, Ant, JCL, ...)
 - Programme (z.B. Java, Cobol, Assembler, ...)
 - Workflow-Beschreibungen (z.B. BPEL, Wf-Engine-Sprachen,...)
 - Schaltungen (z.B. VHDL, Verilog, System/C).
 - . . .
- **Die Anwendungsgebiete sind so zahlreich, dass sich eher eine Gruppierung nach Lebensphase als nach Zielsprache anbietet.**

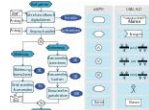
©2005, Dr. Harald Störrie

Anwendungsbereiche für Aktivitätsdiagramme

Algorithmische Abläufe



EPK



Geschäfts-/ Softwareprozess



Build-/Batch-Prozesse



kontinuierliche Prozesse

?

Dienstkomposition



©2005, Dr. Harald Störrie

Anwendungsbereiche für Aktivitätsdiagramme

- **Analyse**
 - Beschreibung existierender oder geplanter Abläufe in der Anwendungsdomäne (Geschäfts- und Softwareprozesse).
- **Entwurf**
 - Komposition von Abläufen aus vorgefertigten Elementen (insbesondere für serviceorientierte Architekturen).
- **Implementation**
 - Dokumentation existierender Programme, Dienste, oder Funktionen
 - Entwurf algorithmischer Abläufe in Implementierungssprachen
 - Schaltungen



©2005, Dr. Harald Störrie

Anwendungsbereiche für Aktivitätsdiagramme

- **Aktivitätsdiagramme können gleichermaßen für die Modellierung geplanter wie existierender Systeme verwendet werden.**
- **Bei geeigneter aufgabenbezogener Auswahl von Ausdrucksmitteln werden Aktivitätsdiagramme problemlos von Nicht-Informatikern akzeptiert.**
- **Im Gegensatz zu Spezialnotationen (z.B. EPKs, DSLs) bieten Aktivitätsdiagramme die üblichen Vorteile der Standardisierung, Integration und Marktberreinigung.**
- **Die Nachteile (v.a. Umgewöhnung, bislang schwache Werkzeugunterstützung) wirken nur verzögernd, nicht verhindernd.**
- **Aktivitätsdiagramme werden für Verhaltensbeschreibungen das sein, was Klassendiagramme für Strukturbeschreibungen heute schon sind.**

©2005, Dr. Harald Störrie

Aktivitätsdiagramme für algorithmische Abläufe

- **Zielsprache Java**
 - Strukturierte Knoten im Sinne von Nassi-Shneiderman-Diagrammen direkt interpretierbar
 - Java-Ausdrücke als Anschriftensprache (z.B. für Transformations- und Selektionsregeln)
- **Zielsprache WSDL**
 - Nur die Schnittstellen werden extrahiert
- **Zielsprache SQL-Queries**
 - spezielle Aktionen für vordefinierte Sprachelemente (Joins, Selects etc.)
 - vor allem Datenfluss-Konstrukte

©2005, Dr. Harald Störrie

Aktivitätsdiagramme für Prozesse

- **Geschäftsprozesse**
 - Es fehlen verschiedene Ausdrucksmittel und Algorithmen
 - **quantitative Aspekte (Dauer, Kosten, ...)**
 - **Transaktionskonzept (auf Basis von Ausnahmen?)**
 - **Entscheidungstabellen zur Abbildung von komplexen Geschäftsregeln**
 - **sequentiell beliebige Ausführung**
 - Anleihen bei EPKs bzw. EPK-Erweiterungen
- **Softwareprozesse**
 - Partitionen müssen zu Rollen weiterentwickelt werden (unterschiedlichen Aufgaben, z.B. ausführend vs. beratend)
 - Ebenfalls: quantitative Aspekte (hybride Aktivitätsdiagramme?)
 - Schleifenknoten werden nicht benötigt, aber Stromverarbeitung ist wichtig (Kanban-Prozess).
 - Es fehlen n-aus-m-Knoten (Spezialfall von fork/join?)

©2005, Dr. Harald Störrie

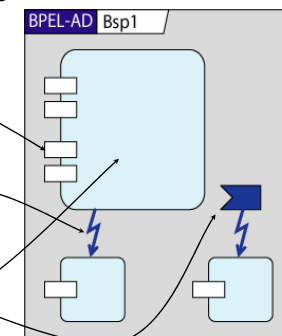
Zielsprache BPEL

Aufbau eines BPEL-Prozesses

```
<process name = "Bsp1" ... >
```

```
PARTNER LINKS
PARTNERS
VARIABLES
CORRELATION SETS
FAULT HANDLERS
COMPENSATION HANDLERS
EVENT HANDLERS
ACTIVITY
```

```
</process>
```




©2005, Dr. Harald Störrie

BPEL vs. UML AD

• Ereignisse

- receive 
- reply 
- invoke 
- wait 
- pick 







• Datenfluss

- assign 

• Abstraktion

- scope 

• Kontrollfluss

- flow 
- sequence 
- switch 
- while 
- Ausnahmen
 - throw 
 - compensate 

©2005, Dr. Harald Störrle

Gliederung

- Einleitung
- Syntax
- Semantik
 - Probleme
- Pragmatik
- Zusammenfassung
- Ausblick

©2005, Dr. Harald Störrle

Fragen für den praktischen Einsatz

- Wozu kann man, wozu soll man UML ADs verwenden?
- Welche Beziehungen bestehen zwischen Aktivitätsdiagrammen und anderen Modellen?
- Welche Ausdrucksmittel fehlen noch für welche Anwendung?
- Welche Werkzeugunterstützung ist jeweils erforderlich?

©2005, Dr. Harald Störrle

Zusammenfassung & Ausblick

- Aktivitätsdiagramme in UML 2 sind mächtig und komplex – vielleicht zu komplex.
- Eine Petrinetz-Semantik für die einzelnen Aspekte liegt vor, es gibt aber eine Reihe von offenen Fragen – nicht zuletzt als Ergebnis der Formalisierung.
- Andere Semantiken sind im Wesentlichen noch nicht vorgeschlagen worden.
- Wahrscheinlich kann man in einer einheitlichen Semantik nicht alle Fragen gleichermaßen gut klären. Womöglich müssen wir also verschiedene, nach Verwendungsarten differenzierte Semantiken definieren (anders als bei StateMachines und Interactions).
- Aber welche Verwendungsarten sind ausschlaggebend? Welche Notationselemente sollten jeweils erlaubt sein, welche fehlen noch? Wie sollte das gleiche Konstrukt (z.B. Ausnahmen) jeweils definiert werden?
- Gibt es vielleicht doch noch einen anderen Ausweg?

©2005, Dr. Harald Störrle

Nächste Schritte

- **Operationale Deutung**
 - Werkzeug-gestützte Analyse von Eigenschaften
 - Anwendungsspezifische Semantiken (BPEL4WS, Java, ...)
- **Deklarative (lose) Semantik analog Petrinetz-Prozess**
- **Praktische**
 - Methodische Einbettung
 - Ausdrucksfähigkeit für bestimmte Anwendungen ergänzen (z.B. Transaktionen)
- **Systematischer Vergleich der Ausdrucksmöglichkeiten von ADs, BPEL4WS und EPKs**

©2005, Dr. Harald Störrie

Weitere Arbeiten zum Thema (2)

- **Semantics of Control Flow in UML 2.0 Activities**
 - IEEE Symposium Visual Languages & Human-Centric Computing (10/2004)
- **Semantics and Verification of Data Flow in UML 2.0 Activities**
 - Intl. Ws. Visual Languages and Formal Methods (VLFM'04)
- **Semantics of Exceptions in UML 2.0 Activities**
 - Journal of Software and Systems Modeling (11/2004)
- **Semantics of Expansion Regions in UML 2.0 Activities**
 - Nordic Workshop on UML (8/2004)
- Verfügbar unter www.pst.ifi.lmu.de/~stoerrie

©2005, Dr. Harald Störrie

Weitere Arbeiten zum Thema (1)

- **Tutorials**
 - UML 2 Concepts and Semantics SEFM'05, 5.9. in Koblenz
 - UML 2 Introduction VL/HCC'05, 21.9. in Dallas
 - UML 2 Aktivitätsdiagramme VSEK, siehe www.vsek.org
- **Bücher**
 - UML 2 erfolgreich einsetzen Addison-Wesley, 2005
 - UML 2 für Studenten Pearson Deutschland, 2005
- **Artikel**
 - www.pst.ifi.lmu.de/~stoerrie

©2005, Dr. Harald Störrie

Aktivitätsdiagramme für Schaltungen

- **Zielsprache VHDL**
 - Ausgewählte Bibliothekskomponenten als Aktivitäten modellieren, z.B. eine ALU
- Datenfluss als „Verdrahtung“



©2005, Dr. Harald Störrie