

Hauptseminar

Semantik der UML2.0 Aktivitäten

von Tong Tong

Gliederung

1. [Einleitung](#)
 - 1.1 [Version UML1.1, UML 1.3 und UML2.0](#)
 - 1.2 [Motivation der Aktivitätsdiagramme](#)
 - 1.3 [Definition der Aktivitätsdiagramme](#)
2. [Die Syntax der Aktivitätsdiagramme](#)
 - 2.1 [Abstrakte Syntax](#)
 - 2.1.1 [Die Struktur von sechs Ebenen](#)
 - 2.1.2 [Zwei grundlegende Konstrukte: Aktion und Aktivitäten](#)
 - 2.1.3 [BasicActivities](#)
 - 2.1.4 [IntermediateActivities](#)
 - 2.2 [Konkrete Syntax](#)
 - 2.2.1 [Aktivitätsknoten](#)
 - 2.2.2 [Aktivitätskanten](#)
 - 2.2.3 [Partitionen](#)
 - 2.2.4 [CallBehaviorAktion](#)
 - 2.3 [Ein Beispiel](#)
3. [Semantik der Aktivitätsdiagramme](#)
 - 3.1 [Grundlegende Definition von Petri-Netzen](#)
 - 3.1.1 [Struktur Definition](#)
 - 3.1.2 [Verhalten Definition von Vollständige Petri-Netzen](#)
 - 3.1.3 [Definition von Prozeduralen Petri-Netzen](#)
 - 3.2 [Semantik der Kontrollflüsse](#)
 - 3.2.1 [Semantik Domäne](#)
 - 3.2.2 [Semantik Abbildung](#)
 - 3.2.3 [Semantik vom Beispiel in Petri-Netzen](#)
 - 3.3 [Semantik der Aufruf Aktivitäten](#)
 - 3.3.1 [Semantik Domäne und Abbildung](#)
 - 3.3.2 [Semantik vom Beispiel in Petri-Netzen mit Verhalten](#)
4. [Konklusion](#)
 - 4.1 [Aktivitätsdiagramme in UML1.x](#)
 - 4.2 [Zusammenfassung](#)

[Literatur](#)

1. Einleitung

Komplexe Software erfordert bereits auf hoher Ebene explizites Design, das in effizienter Weise zu einer zuverlässigen Implementierung weiterentwickelt werden kann. Neben den vielen Sprachen, die benutzt werden, um Software zu programmieren, gibt es noch Modellierungssprachen, die dazu dienen, Programme zu designen und zu analysieren. Zu diesen Modellierungssprachen zählt UML (Unified Modelling Language), die in der Version 2.0 das Thema dieses Seminars ist.

Die Unified Modeling Language (UML) ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme. [Oes04]

1.1 Version UML1.1, UML1.3 und UML2.0

1997 wurde UML der OMG (Object Management Group) zur Standardisierung angeboten. In diesem Jahr stießen auch weitere Firmen dazu, die auf die Anfragen der OMG in diese Richtung hin gearbeitet hatten. Dazu gehörten Firmen wie Ericsson, ObjecTime Limited, Platinum Technology, Softteam und viele andere. Dadurch kam es zu einer neuerlichen Erweiterung bis schließlich die Version 1.1 der OMG vorgelegt wurde. Seit diesem Zeitpunkt übernahm die OMG Revision Task Force (RTF) der weiteren Entwicklung von UML. Im Jahr 1999, erscheint die Version 1.3. In UML1.3 gibt es viele Änderungen über Use-Case- und Aktivitätsdiagramm. Der Standard für 2.0 wurde im November 2004 erst verabschiedet. UML hauptsächlich zwei Diagramme Typen: Strukturdiagramme und Verhaltensdiagramme. Abb1-1 zeigt die darauf abgestützten wesentlichen UML-Diagramme.

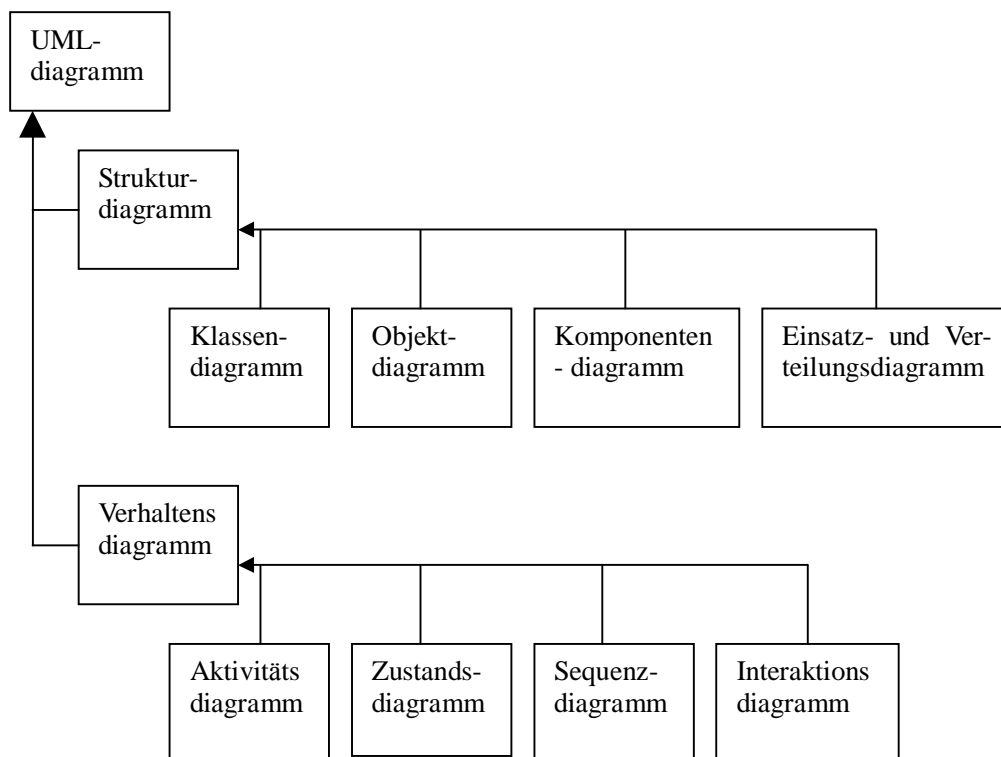


Abb. 1-1

In dieser Ausarbeitung werden nur Aktivitätsdiagramme diskutiert.

1.2 Motivation

Mit Aktivitätsdiagrammen können Abläufe, die beispielsweise in natürlicher Sprache beschrieben sind, grafisch dargestellt werden. Mit natürlicher Sprache können gewöhnlich nur sehr einfache Abläufe verständlich beschrieben werden, während es mit Aktivitätsdiagrammen es möglich ist, auch sehr komplexe Abläufe mit vielen Ausnahmen, Varianten, Sprüngen und Wiederholungen übersichtlich und verständlich

darzustellen. Arbeitsabläufe werden im Software-Engineering mit Aktivitätsdiagrammen modelliert. Deshalb werden Aktivitätsdiagramme als Sprache zur Definition von Abläufen genannt.

1.3 Die Definition der Aktivitätsdiagramme

Ein Aktivitätsdiagramm beschreibt einen Ablauf und wird definiert durch verschiedene Arten von Knoten, die durch Aktivitätskanten mit einander verbunden sind[Oes04]. Es besteht aus Objektknoten, Kontrollknoten, eine Reihe von Aktionen, die gehören zu einige Objekt können.

Wie sieht eigentlich das Aktivitätsdiagramm in UML2.0 aus? Anfangen wir einfach mit der Syntax an.

2. Die Syntax der Aktivitätsdiagramme

Die Syntax der Aktivitätsdiagramme kann von zwei Gesichtspunkte, als „abstrakte Syntax“ und als „konkrete Syntax“, betrachtet werden.

2.1 Abstrakte Syntax

2.1.1 Die Strukturierung der Beschreibungsmittel in sechs Gruppen

In UML2.0 werden Aktivitätsdiagramme mit Hilfe von sechs Gruppen von Beschreibungsmitteln definiert. Abb2-2 zeigt uns die Struktur dieser sechs Gruppen. *BasicActivities* definiert den traditionellen Kontrollfluß und den Übergang zu Kontrollflußverfeinerungen (prozeduraler Aufruf von untergeordneten Aktivitäten). *IntermediateActivities* erweitert und präzisiert die Definition von Kontrollflüssen und bezieht Nebenläufigkeiten und Datenflüsse mit ein. Es basiert auf *BasicActivities*. *CompleteActivities* stellt als Erweiterung von *IntermediateActivities* zusätzliche Beschreibungsmittel bereit. *StructuredActivities* führt traditionelle Strukturierungen (wie Schleifen, Auswahl unter Bedingungen, etc.) ein. Es basiert auf *BasicActivities*. *StructuredActivities* und *IntermediateActivities* sind orthogonal. Das heißt, ihre Beschreibungsmittel können unabhängig voneinander (gegebenenfalls gemeinsam) benutzt werden. *ExtraStructuredActivities* und *CompleteStructuredActivities* sind Erweiterungen von *StructuredActivities*. *CompleteStructuredActivities* fügt Strukturelemente (wie Schleifen und Bedingungen) für Datenflüsse hinzu. *ExtraStructuredActivities* erlaubt die Formulierung von Ausnahmen (Exceptions).

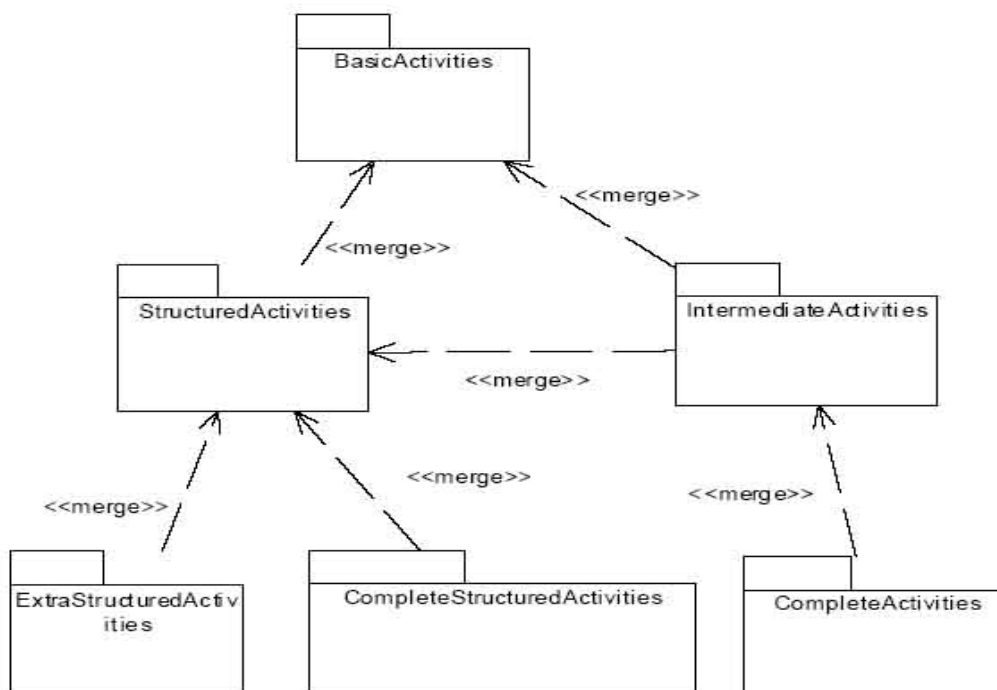


Abb. 2-2

2.1.2 Zwei grundlegende Konstrukte

In *BasicActivities* gibt es zwei grundlegende Konstrukte: Aktionen und Aktivitäten.

Aktionen sind die fundamentale Einheit für die Beschreibung ausführbarer Funktionalität[Stö04]. Aktionen können, auf der gegebenen Abstraktionsebene, nicht weiter zerlegt werden. Sie können allerdings zu Aktivitäten zusammengefasst werden. Jede Aktion kann in einer Aktivität ein- oder mehrmals, gegebenenfalls auch nicht ausgeführt werden. Aktionen verfügen über den Zugriff auf Daten, die sie transformieren und/oder testen können.

Aktivitäten koordinieren die Ausführungsreihenfolge von Aktionen und (Sub-) Aktivitäten [Stö04]. Die Koordination des Ablaufes wird durch einen Graphen dargestellt, in dem „Aktivitätsknoten“ durch „Aktivitätskanten“ verbunden werden. Aktionen können nicht mehr verfeinert werden, aber Aktivitäten können andere Aktivitäten enthalten, die, durch die eine Verfeinerung des Verhaltens beschrieben wird. In einem vollständigen Diagramm sind alle Aktivitäten auf Aktionen heruntergebrochen.

Abb2-1[Stö04] zeigt uns die Beziehung zwischen Aktionen und Aktivitäten.

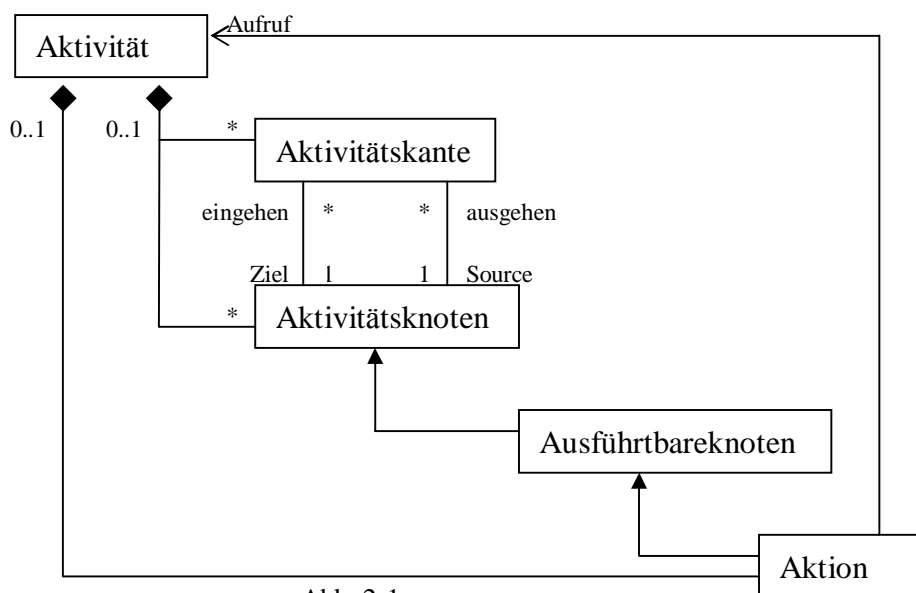


Abb. 2-1

Abb. 2-1 verlangt also, dass eine Aktivität aus eine Aktion oder eine Abbildung von Aktivitätsknoten und Aktivitätskanten bestehen. Eine Aktion ist eine Vererbung von Ausführbareknoten. Deswegen können wir die Aktivität von einer Form aufzeigen:

<Name, <Aktivitätsknoten, Aktivitätskanten>>

2.1.3 BasicActivities

In *BasicActivities* wird die Struktur von Knoten durch Abb2-3 gezeigt.

Ein Aktivitätsknoten ist der allgemeine Name von Knoten. Ein Ausführbareknoten ist der allgemeine Name von Aktionen. Ein Objektknoten gibt an, dass ein Objekt oder eine Menge von Objekten existiert. Es kann durch so genannte Pins dargestellt werden oder als ein- und ausgehende Parameterknoten in Aktivitäten verwendet werden. Ein Kontrollknoten ist einen Controller von Abläufen. Es gibt Startknoten(Anfangsknoten), der ein Startpunkt eines Ablaufes ist, und Endknoten, der den beschriebenen Ablauf beendet. Ein Zusammenführungsknoten ist eine Oder-Verknüpfung. Das heißt, bei diesen Knoten führt jeder von mehreren eingehenden Kontrollflüsse sofort zu einem gemeinsamen ausgehende Kontrollfluss. Ein Entscheidungsknoten ist eine Verzweigung mit einem oder mehr ausgehenden Flüsse, die aufgrund von Bedingungen entscheiden wird. Auf diese Ebene wird nur grundlegendes Verhältnis von

Entscheidungsknoten erklärt.

Aktivitätskanten bestehen aus zwei Typen: Kontrollflüsse und Objektflüsse. Kontrollflüsse sind die Verbindungen zwischen Ausführbareknoten oder Kontrollknoten. Objektflüsse sind nicht nur die Verbindungen für Kontrollieren, sondern auch den Transport für Daten. Auf diese Ebene wird kein Objektfluss definiert.

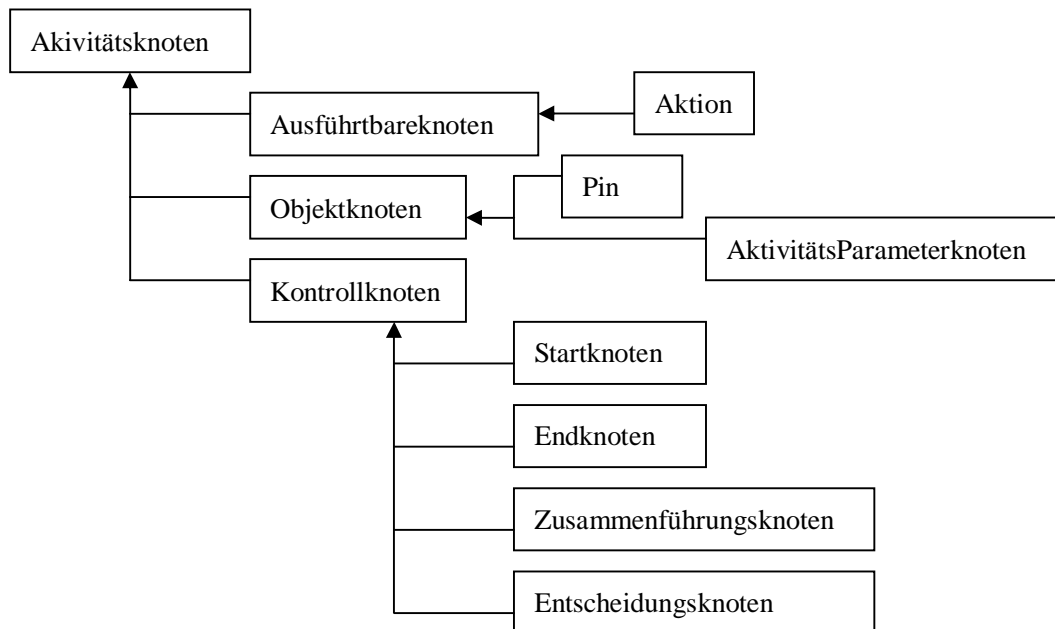


Abb. 2-3

2.1.4 IntermediateActivities

In *IntermediateActivities* werden zuerst einige neue Kontrollknoten hinzugefügt. Diesen Knoten werden in Abb. 2-4 durch drei helle graue Rechtecke gezeigt. Ein Teilungsknoten ist ein Schritt im Ablauf, an dem ein eingehender Kontrollfluss ohne Bedingungen sofort in mehrere ausgehende nebenläufige Kontrollflüsse geteilt wird. Ein Synchronisationsknoten ist eine UND-Verknüpfung. Das heißt, bei diesen Knoten müssen alle eingehende Flüsse warten, bevor der Kontrollfluss fortgesetzt wird. Ablaufende ist ein Endknoten, der einen einzelnen Kontrollfluss beendet.

Bei Objektknoten wird ein neuer Knoten definiert. Das heißt CentralBufferKnoten. Es ist ein Bereich, der Daten speichern kann. In Abb. 2-4 wird durch ein dunkles graues Rechteck dargestellt. Die Objektknoten und Objektflüsse werden auf diese Ebene ausführlich definiert. Aber in dieser Ausarbeitung werden nicht erklärt.

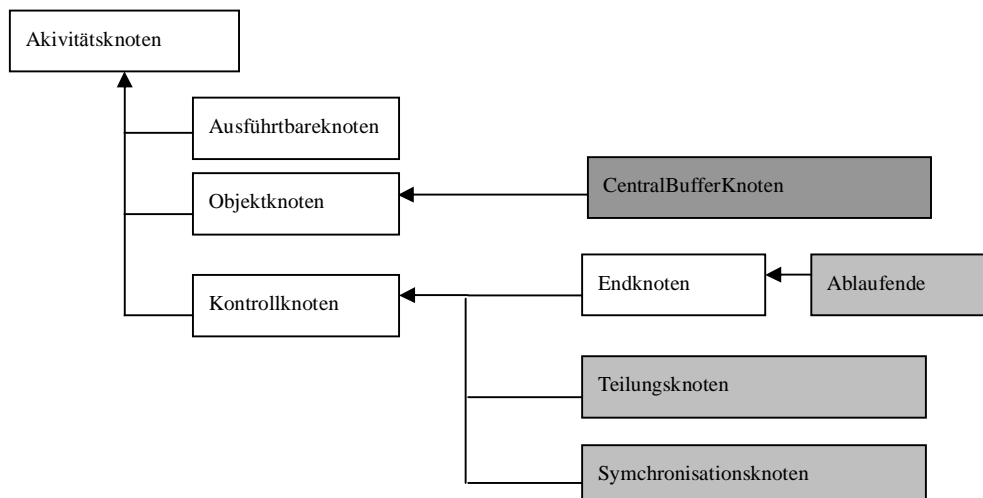


Abb. 2-4

In *IntermediateActivities* wird noch ein neuer Begriff definiert: Partitionen. Eine Partition beschreibt innerhalb eines Aktivitätsmodells, wer oder was für einen Knoten verantwortlich ist oder welche gemeinsame Eigenschaft sie kennzeichnet[Oes04].

Bisher haben wir schon viele Metamodelle eingeführt. Wie werden sie in UML2.0 dargestellt? Um die Frage zu antworten, sollen wir die konkrete Syntax erklären.

2.2 Konkrete Syntax

Es gibt zu viele Konkrete Syntax für Aktivitätsdiagramme. Hier werden nur die grundlegenden Elemente eingeführt.

2.2.1 Ausführbareknoten

Zuerst wird Ausführbareknoten erklärt. Wie Abb. 2-5 zeigt, wird eine Aktion durch ein rundeckiges Rechteck dargestellt. Der Name von einer Aktion, der ein Verhalten sein muss, wird in diesem Symbol geschrieben.

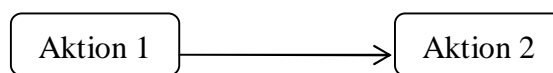


Abb. 2-5

Zweite wird in Abb2-6 Objektknoten gezeigt. Pins sind kleine, am äußeren Rand eines Ausführbareknotens anliegende Quadrate. Alternativ können Objektknoten durch Rechtecke dargestellt werden, die den Namen des Objektes und Optional in eckige Klammern den Objektzustand enthalten.

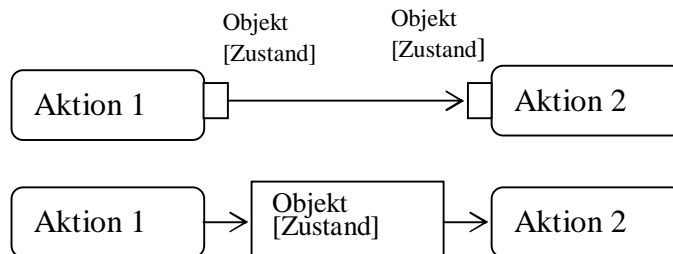


Abb. 2-6

Dritte wird Kontrollknoten eingeführt. Ein Startknoten wird durch einen ausgefüllten Kreis dargestellt. Ein Endknoten hat zusätzlich einen äußeren Ring. Eine Ablaufende wird dargestellt durch einen nicht ausgefüllten gekreuzten Kreis.

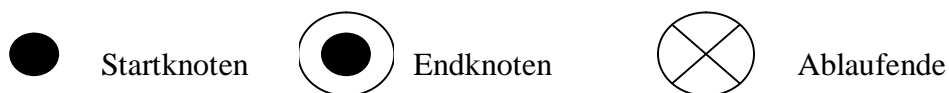
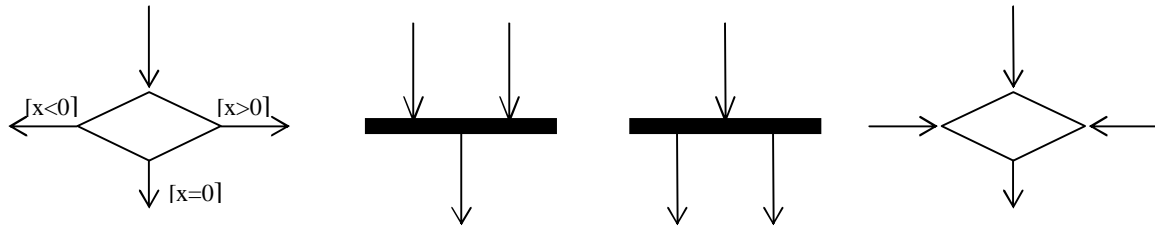


Abb. 2-7

Ein Entscheidungsknoten wird durch eine nicht ausgefüllte Raute dargestellt, die einen eingehende und mehrere ausgehende Kontrollflüsse hat. Ein Synchronisationsknoten wird durch einen Balken repräsentiert, der mehrer eingehende und einen ausgehende Kontrollfluss hat. Ein Teilungsknoten wird ebenfalls durch einen Balken dargestellt, der aber einen eingehende und mehrere ausgehende Kontrollflüsse hat. Ein Zusammenführungsknoten wird dargestellt durch eine nicht ausgefüllte Raute mit mehreren eingehenden und einem ausgehenden Kontrollfluss.



Entscheidungsknoten Synchronisationssknoten Teilungssknoten Zusammenführungsknoten

Abb. 2-8

2.2.2 Aktivitätskanten

Wie obigen Abbildung gezeigt, wird eine Aktivitätskante durch eine Linie mit einer Pfeilspitze dargestellt. Es kann einen Namen haben, der nahe bei der Linie geschrieben wird. Kontrollflüsse können mit Konnektoren versehen werden, um lange, quer durchs Diagramm laufende Linien zu vermeiden oder Kontrollflüsse beispielsweise auf einem anderen Blatt fortzuführen. Ein Konnektor ist ein Kreis, in dem ein eindeutiger Bezeichner steht, beispielsweise ein Buchstabe.

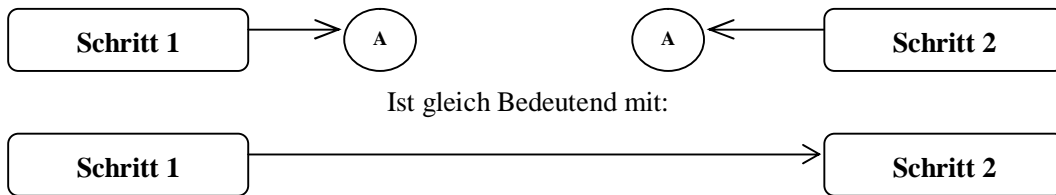
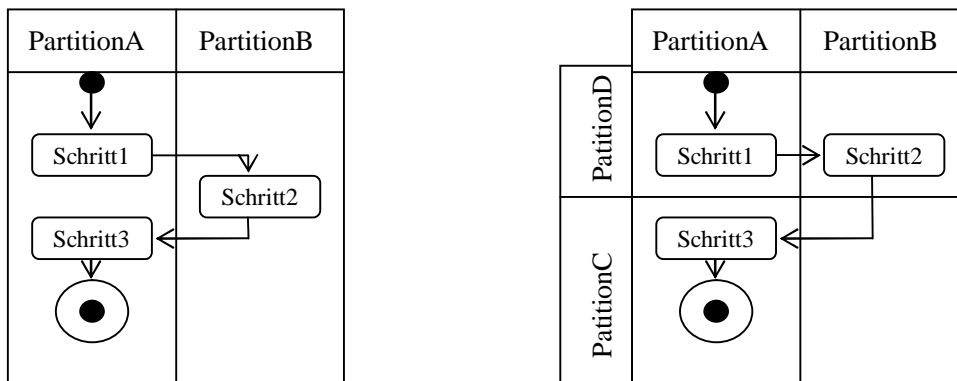


Abb. 2-9

2.2.3 Partitionen

Partitionen können grafisch dargestellt werden durch senkrechte Linien, wodurch das Diagramm in Bahnen aufgeteilt wird. In UML2.0 sind auch mehrdimensionale Partitionen möglich. Das heißt, ein Knoten kann mehreren Partitionen zugeordnet werden. Abb2-10 zeigt uns die zwei Fälle.



Normale Partitionen

Mehrdimensionale Partitionen

Abb. 2-10

2.2.4 CallBehaviorAktion

Wie §2.1.2 gesagt, kann eine Aktivität durch eine Aktion andere Aktivität aufrufen. Diese Aktion besitzt Unteraktionen. Das heißt, eine Aktion kann verschaltet sein und eine komplette Aktivität beinhalten. Wie Abb2-11 können Aktionen mit Unteraktivitätsmodellen durch ein kleines Gabelsymbol innerhalb der Aktion gekennzeichnet werden. Und der Name, die eine aufgerufene Aktivität ist, wird in dieser Aktion geschrieben.

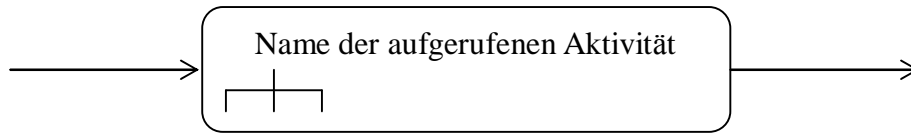


Abb. 2-11

2.3 Ein Beispiel

Hier wird ein Beispiel für ein normales Aktivitätsdiagramm genommen. Durch Abb. 2-12 können wir sehen, dass die Aktivität X und die Aktivität Y. Aktion Y ist der Name von Aktivität Y, und es gibt ein Gabelsymbol innerhalb Aktion Y. Deswegen ruft Aktivität X hier Aktivität Y auf.

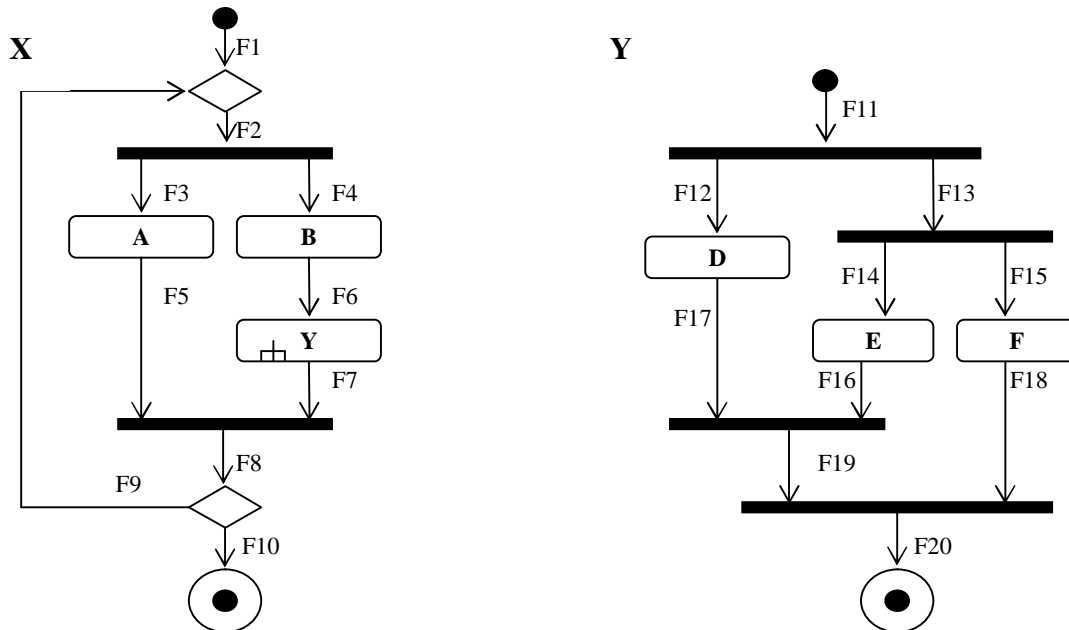


Abb. 2-12

3. Semantik der Aktivitätsdiagramme

In UML2.0 verfügen Aktivitätsdiagramme jetzt über eine Petrinetz-ähnliche Semantik.

Zuerst sollen wir eine kurze Einführung über Petri-Netze machen.

3.1 Grundlegende Definition von Petri-Netzen

Petri-Netze ermöglichen eine Graphen-orientierte Beschreibung verteilter Systeme und deren Abläufen. Formalismus wurde von C.A. Petri 1962 entwickelt. Ansatz führte zu einer Theorie zur Analyse und Entwicklung nebenläufiger System.

3.1.1 Struktur Definition

Wir sollen zuerst die Charakterisierung einführen. Ein Petri-Netz ist gerichteter Graph mit Kanten und zweierlei Knoten.

- | Knoten sind Stellen (graphische Kreis) und Transitionen (graphische Rechtecke).
- | Kanten sind eine gerichtete Verbindung von Stellen zu Transitionen oder von Transitionen zu Stellen.
- | Die Stellen können mit Marken/Werten(Token) belegt werden. In einem booleschen Netz sind als Werte nur 0 oder 1 zugelassen. In einem Stellen/Transitionsnetz ist für die Belegung definiert die Stellenkapazität.
- | Zustand wird durch Belegung der Stellen definiert. Zustandübergang wird durch so genannt Schaltregel geändert.

(In Struktur Definition beziehen wir uns nur auf statische Petri-Netze. Die Schalregel werden wir in

Verhalten Definition eingeführt.)

- Markierung der Kanten heißt Kantengewichte. Gewichtung gibt an, wie viel Marken beim Schalten einer Transition von den Eingangsknoten (Stellen) der Transition abgezogen und den Ausgangsknoten (Stellen) der Transition hinzugefügt werden.

Beispiel:

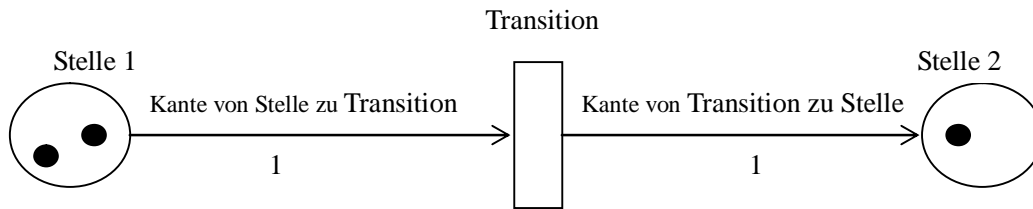


Abb. 3-1

Definition:

Ein Petri-Netz ist ein Tripel $\langle P, T, A \rangle$ mit:

- P ist eine endlich Menge von Stellen.
- T ist eine endlich Menge von Transitionen und es gilt: $P \cap T = \emptyset$ d.h. Stellen und Transitionen sind disjunkt.
- $A \subseteq P \times T \cup T \times P$ ist eine endlich Menge von Kanten.
- Für Netz-Elemente $X = P \cup T$ gilt: $x \in X$,
 $\cdot x = \{ y \mid \langle y, x \rangle \in A \}$ den Vorbereich von x
 $x \cdot = \{ y \mid \langle x, y \rangle \in A \}$ den Nachbereich von x

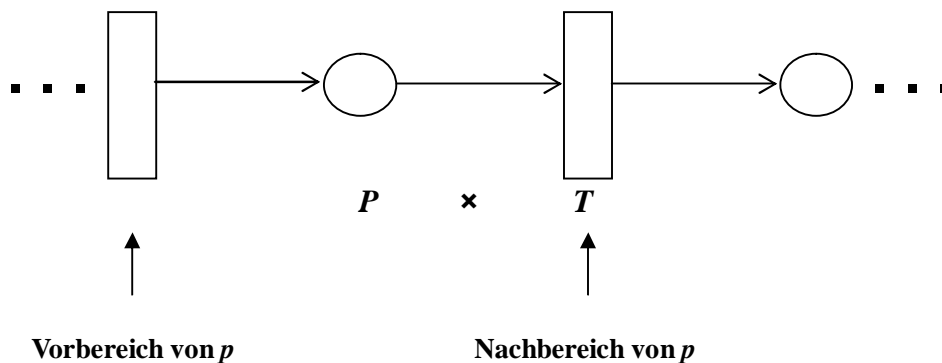


Abb. 3-2

- Markierung:

Eine Abbildung $c: S \rightarrow \mathbb{N} \cup \{ \infty \}$ gibt die Kapazität einer Stelle an. Das heißt, die Kapazität einer Stelle kann Natürlichzahl und unendlich groß sein.

Eine Abbildung $w: F \rightarrow \mathbb{N} \cup \{ 0 \}$ gibt die Gewichtung einer Kante an. Das heißt, die Gewichtung einer Kante kann Natürlichzahl und 0 sein.

Eine Abbildung $M: S \rightarrow \mathbb{N}$ heißt natürliczhahlige Markierung der Stellen. Man kann mit $m(p)$ die Anzahl der Tokens von P zeichnen.

- Ein Quintupel $N = \langle P, T, A, \bar{m}, \underline{m} \rangle$ ist ein Vollständiges Petri-Netz, wenn $\langle P, T, A \rangle$ ein Petri-Netz ist, \bar{m} die Startmarkierung ist, und \underline{m} die Endmarkierung ist.

Mit obiger Definition ist die statische Struktur eines Netzes formal erfasst.

3.1.2 Verhalten Definition von Vollständigen Petri-Netzen

Zur Erfassung des dynamischen Verhaltens erweitern wir die Definition eines Vollständigen Petri-Netzes als die Schaltregel.

Gegeben sei ein Vollständiges Petri-Netz, $N = \langle P, T, A, \bar{m}, \underline{m} \rangle$ und die Markierung m von einer Stelle p mit einem aktivierten Transition t . Man kann das mit $N : m \xrightarrow{t}$ oder $m \xrightarrow{t}$ zeichnen, wenn N klar ist.

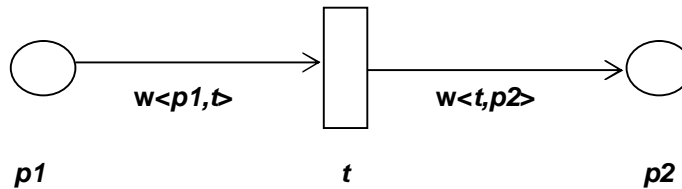


Abb. 3-3

Wie Abb. 3-3, für $p1$ ist Vorbereich von t und $p2$ ist Nachbereich von t . t kann schalten, wenn:

$$m(p1) \geq w(\langle p1, t \rangle)$$

$$w(\langle t, p2 \rangle) + m(p2) \leq c(p2)$$

Ein Zustandübergang erfolgt durch das Schalten von Transition(wie Abb. 3-4), wenn folgende Bedingungen erfüllt sind:

$$m(p) \geq w(\langle p, t \rangle) \text{ für alle } p \in \cdot t$$

$$c(q) \geq w(\langle t, q \rangle) + m(q) \text{ für alle } q \in t \cdot$$

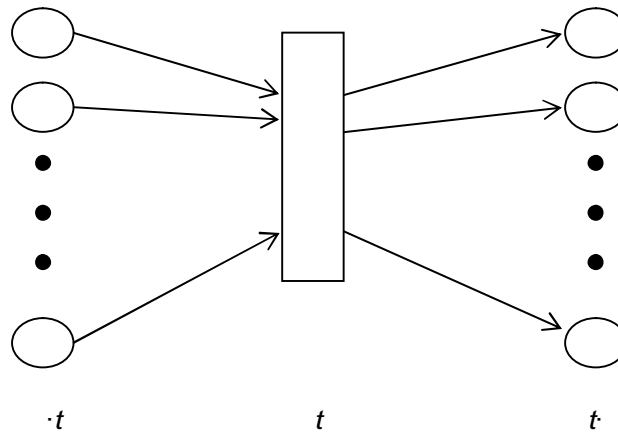


Abb. 3-4

Durch das Schalten von t wird eine Folgemarkierung m' zu m erzeugt. Das kann durch $N : m \xrightarrow{t} m'$ gezeichnet werden.

$$m'(p) = m(p) - w(\langle p, t \rangle) \quad \text{für alle } p \in \cdot t$$

$$m'(q) = m(q) + w(\langle t, q \rangle) \quad \text{für alle } q \in t \cdot$$

Eine Reihe ω von Transitionen(die Länge wird durch $|\omega|$ gezeichnet) kann als Schaltfolge von N mit Startmarkierung \bar{m} genannt, wenn es $\{m_0, \dots, m_{|\omega|}\}$ gibt, und $\forall 0 < i < |\omega| : m_{i-1} \xrightarrow{\omega_i} m_i$ • (Abkürzung: $m_0 \xrightarrow{\omega} m_{|\omega|}$ •). Davon besteht die Folge ω aus ω_i .

Eine Folgemarkierung m' ist erreichbar von einer Markierung m ($m \xrightarrow{\omega} m'$), wenn es eine feuernde Reihe ω gibt. Und die erreichbare Markierung von m kann durch $m \xrightarrow{\omega}$ gezeichnet.

Wenn alle Transition schalten, können wir eine Endmarkierung \underline{m} erreichen.

$$N : \bar{m} = m_0 \xrightarrow{\omega} m_0' = m_1 \xrightarrow{\omega} m_1' = \dots \xrightarrow{\omega} m_n = \underline{m}$$

3.1.3 Definition von Prozeduralen Petri-Netzen

Prozedurale Petri-Netze (PPN) ist ein einfacher Mechanismus, um zusammenschließenden prozeduralen Aufruf-Mechanismus in grundlegende Netze zu verwenden. In dieser Ausarbeitung werden nur die einfachen Definitionen erklärt.

Ø Definition für die Struktur von PPN:

Ein Paar $NS = \langle N, \rho \rangle$ ist ein PPN, wenn N eine endliche Menge von Vollständige Petri-Netze ist, von der jede eine zusätzliche partielle Funktion ($act_N: T_N \rightarrow A$) hat, um den Name von der aufgerufenen Aktivität in Transition anzuweisen. Wenn keine Aktivität aufgerufen wird, wird act nicht definiert. $\rho: T_N \rightarrow N$ ist eine partielle Funktion für Aufruf. Durch diesen Funktion weist eine Transition in diese Menge von Petri-Netze die aufgerufene Petri-Netze auf. Das heißt, ein Petri-Netz kann durch eine Transition die anderen Petri-Netze aufrufen. Hier N ist ein Element von N , und $T_N = \bigcup_{N \in N} T_N$.

Die Zustand von NS sind Elemente von $C \times I \times M \times F$:

- | C enthält die Aufruf-Transitionen. $\text{dom}(\rho) \cup \{\perp\}$.
- | I ist eine endliche Reihe der eindeutigen Instanzidentifikatoren.
- | M ist das Klass von Markierung der Netzte in NS.
- | F ist eine Reihe von aktuellen Aufrufen.

Ein Zustand Element in einem Netz wird als den Aufrufer mit \perp gezeichnet, wenn das Netz nicht aufgerufen werden. \perp kann die Wurzel von einem Baum zeichnen, auch kann die Abhängigkeit von Aufruf zeichnen.

Ø Definition für normale Transitionen:

Transition t ist nicht gehört zu $\text{dom}(\rho)$. Das heißt, t sind keine Aufruf-Transitionen. t ist gefeuert in s , wenn $\langle c, i, m, f \rangle \in s$ gehört, und entweder $t \in T_{\rho(c)}$ gehört oder $c = \perp$. Die neue Zustand s' wird durch t erzeugt mit $s' = s - \langle c, i, m, f \rangle + \langle c, i, m', f \rangle$. m' kann durch Verhalten Definition von Petri-Netze gerechnet. Wenn $t \in T_{\rho(c)}$, das heißt, die Transition ist in aufgerufenen Netze. Wenn $c = \perp$, dann ist die Transition in aufrufenden Netze.

Ø Definition für prozedurale Aufruf-Transitionen:

Eine Transition t ($t \in \text{dom}(\rho)$) in Zustand s kann eine prozedurale Aufruf-Transition sein. Sie kann mit $s \xrightarrow{t_{\text{call}}} s'$ gezeichnet werden. Wenn $s \xrightarrow{t_{\text{call}}} s'$ gefeuert ist, werden eine neue Instanz i' und ein neuer Zustand s' erzeugt, und mit $s \xrightarrow{t'_{\text{call}}} s'$ gezeichnet.

Deshalb $s' = s - \langle c, i, m, f \rangle + \langle c, i, m \cdot t, f \cup \{i'\} \rangle + \langle t, i', \overline{m_{\rho(t)}}, \emptyset \rangle$.

Ø Definition für prozedurale Rückkehr-Transitionen:

Eine Transition t in Instanz i' kann eine Rückkehr-Transition zu s sein. Sie kann mit $s \xrightarrow{t'_{\text{return}}} s'$ gezeichnet werden. $s \xrightarrow{t'_{\text{return}}} s'$ ist gefeuert, das heißt, Instanz i' hat Endmarkierung und kein Unter-Aufruf. Das kann mit $\langle c, i, m, f \cup \{i'\} \rangle \in s$ gezeichnet. Durch gefeuertes t'_{return} , wird eine neue Zustand s' erzeugt, und wird die Instanz i' entfernt.

Deshalb $s' = s - \langle c, i, m, f \cup \{i'\} \rangle + \langle c, i, m + t, f \rangle - \langle t, i', \underline{m_{\rho(t)}}, \emptyset \rangle$.

3.2 Semantik der Kontrollflüsse

Die Semantik der grundlegenden Kontrollflüsse wird hier definiert. Wir nehmen an, dass jede Aktivität nur die Knoten hat, die in dieser Ausarbeitung eingeführt werden, und zwar nur einen Endknoten.

3.2.1 Semantik Domäne

Wir definieren hier Domäne A für Aktionen. Die Funktion $call(_)$ wird als Name von CallBehaviorAktion

genannt. Wenn eine Aktion keine CallBehaviorAktion ist, wird als \perp genannt.

Dann definieren wir Domäne PN für Vollständige Petri-Netze mit einem Tupel $\langle P, T, A, \bar{m}, \underline{m} \rangle$. P, T, A haben die normale Bedeutung, während \bar{m} Startmarkierung und \underline{m} Endmarkierung bedeutet.

In §2.1.2 haben wir eine Form von Aktivitäten definiert: $\langle \text{Name}, \langle \text{Aktivitätsknoten}, \text{Aktivitätskanten} \rangle \rangle$

Name ist die Namen von Aktivitäten.

Aktivitätsknoten können in einem Tupel aufgeteilt werden: $\langle \text{EN}, \text{iN}, \text{fN}, \text{BN}, \text{CN} \rangle$

- | EN ist die Ausführbareknoten (Aktionen).
- | iN, fN sind die Startknoten und Endknoten.
- | BN ist die Zusammenführungsknoten und Entscheidungsknoten.
- | CN ist die Teilungsknoten und Synchronisationsknoten.

Aktivitätskanten sind die Verbindung zwischen Aktivitätsknoten.

Aber davon haben wir keine Datenknoten und Datenflüsse. Jetzt können wir die Domäne von Petri-Netze in Aktivitätsdiagrammen verwenden: $\llbracket \langle \text{Aktivitätsknoten}, \text{Aktivitätskanten} \rangle \rrbracket = \langle P, T, A, \bar{m}, \underline{m} \rangle$

Weil wir keine Datenflüsse und Datenknoten besprechen, können wir die obige Domäne in Kontrollflüsse übernehmen.

3.2.2 Semantik Abbildung

Die grundlegende Kontrollflüsse können einfach in drei Teile verteilt werden. Ausführbareknoten werden in Transition übersetzt. Kontrollknoten werden in Stelle übersetzt. Aktivitätskanten werden in Kanten übersetzt. Jetzt sollen wir die Gleichung „ $\llbracket \langle \text{Aktivitätsknoten}, \text{Aktivitätskanten} \rangle \rrbracket = \langle P, T, A, \bar{m}, \underline{m} \rangle$ “ mit die grundlegende Symbol genauer erklärt.

- | $P = \{ \text{iN}, \text{fN} \} \cup \text{BN} \cup \{ p_a \mid a \in \text{Kontrollflüsse}, \{ a_1, a_2 \} \cap (\text{EN} \cup \text{CN}) \neq \emptyset \}$

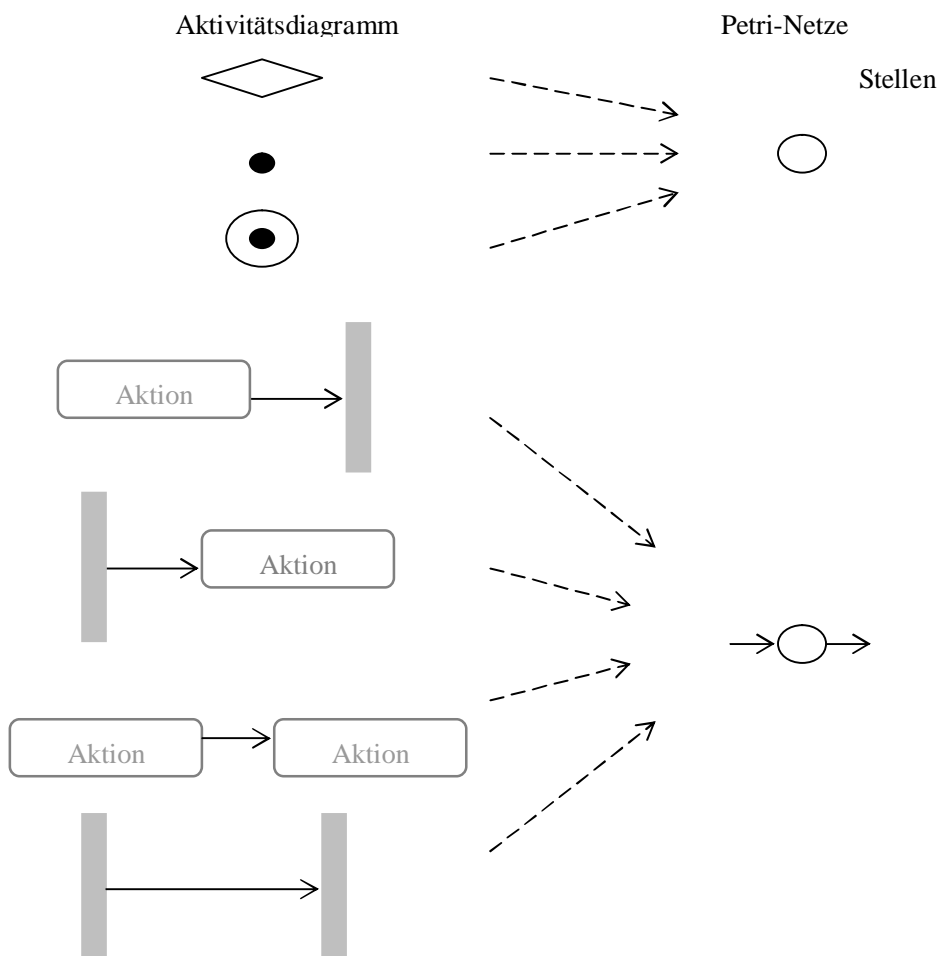


Abb. 3-3

$$T = EN \cup CN \cup \{ t_a \mid a \in \text{Kontrollflüsse}, \{a_1, a_2\} \subseteq BN \cup \{iN, fN\} \}$$

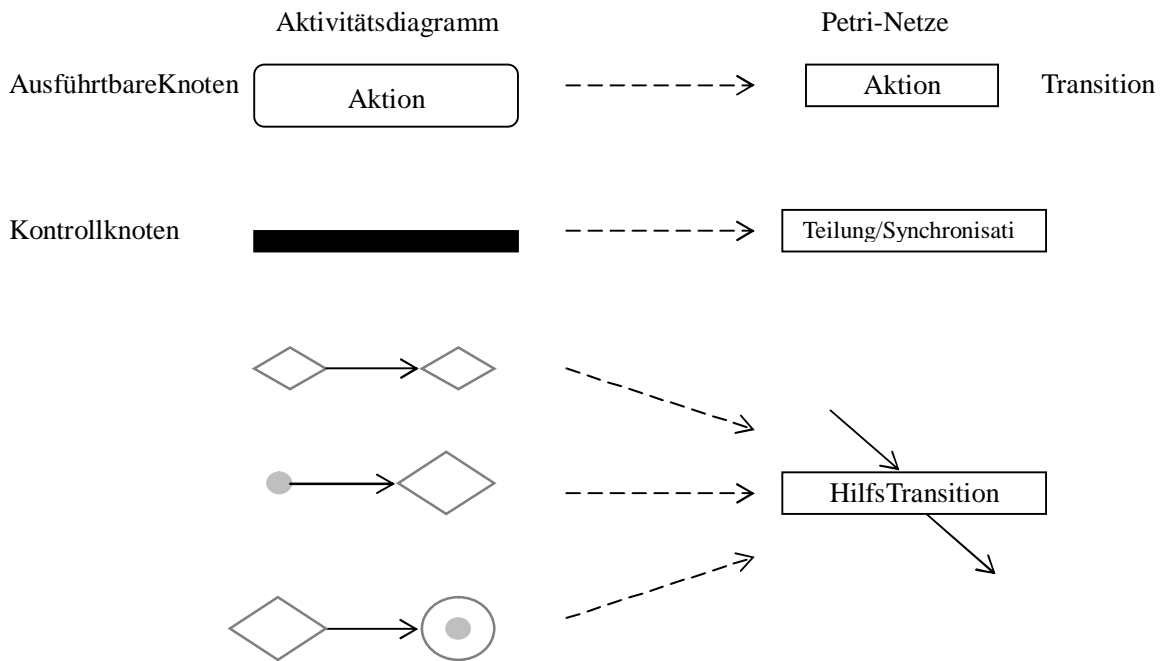


Abb. 3-4

$$A = \{ \langle X_{\langle \text{von}, \text{nach} \rangle}, \text{nach} \rangle, \langle \text{von}, X_{\langle \text{von}, \text{nach} \rangle} \rangle \mid \langle \text{von}, \text{nach} \rangle \in \text{Kontrollkanten} \}$$

Das heißt A ist die Kontrollflüsse, bei der einen Seite ist p_a ($p_a \in P$ und $a \in \text{Kontrollflüsse}$) oder t_a ($t_a \in T$ und $a \in \text{Kontrollflüsse}$).

$$\bar{m} = iN$$

$$\underline{m} = fN$$

3.2.3 Semantik vom Beispiel in Petri-Netzen

Mit obigen Definitionen können wir die Semantik vom Beispiel in § 2.3 darstellen.

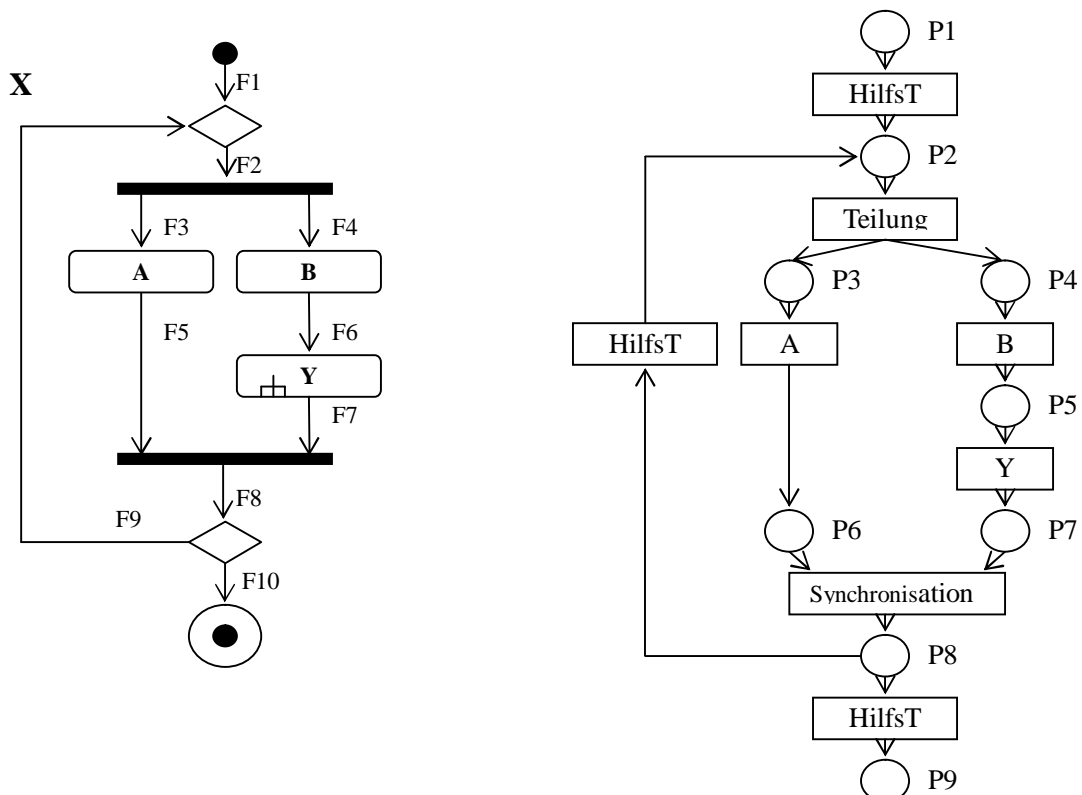


Abb. 3-5

Y

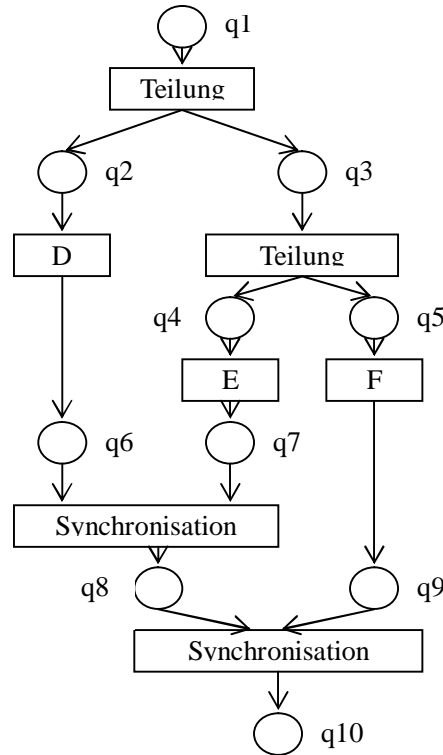
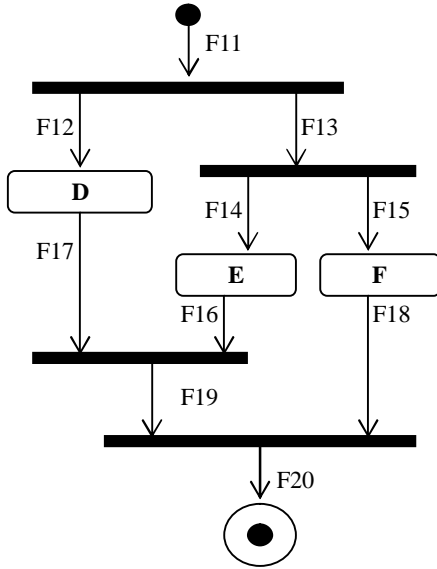


Abb. 3-6

Nach der Übersetzung können wir sehen, dass die CallBehaviorAktion von Aktion Y in Aktivität X nicht dargestellt wird. Das heißt, dass die zwei Aktivitäten jetzt unabhängig sind.

3.3 Semantik der Aufruf Aktivitäten

3.3.1 Semantik Domäne und Abbildung

Um die Beziehung von Aufruf darzustellen, suchen wir nach der Lösung in der Petri-Netze Domäne. Zuerst, können wir das traditionale Netze-Morphismus oder eine statische Erweiterung durch Unternetzen verwenden. Aber wenn viele Aufrufe von demselben Unternetz gleichzeitig passieren, oder zwischen Aufrufe Interaktionen geben, werden die Aufrufe verhindert. Um diese Probleme zu vermeiden, sollen alle Tokens durch Indertifikator sich unterscheiden, oder Höher-Order Netze werden verwendet. Aber es kann zu einem sehr großen Ergebnis führen.

Jetzt sollen wir eine neue Lösung einführen. Das sind Prozedurale Petri-Netze (PPN).

Wir nehmen *Spec* als eine Reihe von Aktivitäten an. Es muss eine Spitzen-Aktivität geben. Das heißt, sie ist die Anfangszustand von der ganzen Reihe. Sie kann durch Funktion $top(_)$ ausgewählt. Die Namen von Aktivitäten sollen unterschied sein, und alle Aktivitäten sind disjunkt. Mit obigen Bedingungen, können wir die Gleichung stellen: $\llbracket Spec \rrbracket = \langle N, \rho \rangle$. Davon N ist $\{\{Diagramm(A)\} \mid A \in Spec\}$, ρ ist eine Reihe von Aufruf-Beziehungen in *Spec*: $\{Name(A) \mathbf{a} [act(A)] \mid A \in Spec, act(a) \text{ ist definiert}\}$.

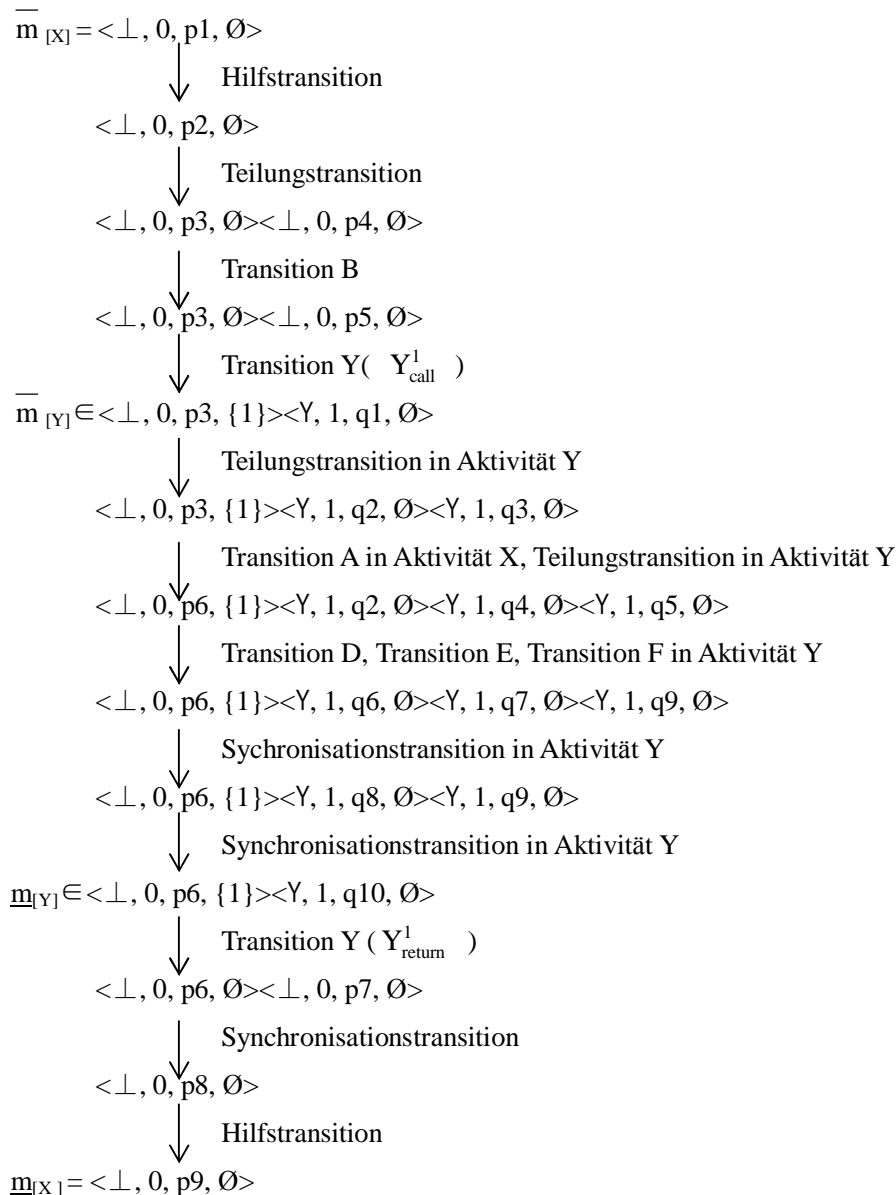
Nehmen wir an, dass in *Spec* nur eine eindeutige Spitzen-Aktivität gibt. Durch $top(Spec)$ wird sie ausgewählt. Und dann die Startzustand ist $\overline{m}_{Spec} = \langle \perp, 0, \overline{m}_{Spec}, \emptyset \rangle$ und die Endzustand ist $\underline{m}_{Spec} = \langle \perp, 0, \underline{m}_{Spec}, \emptyset \rangle$.

3.3.2 Semantik vom Beispiel in Petrinetz mit Verhalten

Das Beispiel, das in §3.2.3 schon in Petri-Netze übersetzt wird, wird hier mit Verhalten beschrieben.

Zuerst, $\llbracket Spec \rrbracket = \langle \{X, Y\}, \{Y \mathbf{a} [Y]\} \rangle$.

Dann $\overline{m}_{spec} = \langle \perp, 0, p1, \emptyset \rangle$ und $\underline{m}_{spec} = \langle \perp, 0, p9, \emptyset \rangle$.



4. Konklusion

Bisher haben wir schon die Aktivitätsdiagramme in UML2.0 kurz eingeführt. In §1.1 beziehen wir uns auf UML1.x. Jetzt können wir genauer schauen, welche Unterschied zwischen alte Versionen(hauptsächlich Version1.3) und UML2.0, und welche Definition in UML2.0 ist Erneuerung von alten Versionen.

4.1 Aktivitätsdiagramme in UML1.x

Aktivitätsdiagramm wird in UML1.x als State Maschine angesehen. Aber jetzt wird nicht mehr. In UML2.0 gibt es keine Relation zwischen Aktivitätsdiagramme und Zustandsdiagramme.

Die einzelnen Schritte im Ablauf heißen in UML1.x Aktivitäten, während in UML2.0 Aktionen.

Jede eingehende Transaktion hat in UML1.x einen Ablaufschritt gestartet, während in UML2.0 alle eingehenden Kontrollflüsse vorliegen müssen, damit die Aktion startet.

Wenn mehrere ausgehende Kontrollflüsse notiert waren, musste in UML1.x über entsprechende Bedingungen sichergestellt werden, dass stets nur ein Kontrollfluss feuern kann. Aber in UML2.0 wird gewartet, bis alle Bedingungen für alle ausgehenden Kontrollflüsse erfüllt sind, bevor gefeuert wird.

Außer obigen Definitionen gibt es auch einige Elemente, die in UML1.x erneuert werden. [Oes04]

- | Aktivitäten können Objektknoten als Ein und Ausgangsparameter haben.
- | Es können Vor- und Nachbedingungen für Aktivitäten definiert werden.
- | Anfangs- und Endzustand heißen jetzt ‚Startknoten‘ und ‚Endknoten‘. Zusätzlich gibt es jetzt das ‚Ablaufende‘.
- | Partitionen(Schwimmbahnen in UML1.x) können mehrdimensional sein, jede Aktion kann mehreren Partitionen zugeordnet werden.
- | Es können unterbrechbar Bereiche definiert werden, die asynchron durch ein eintreffendes Signal verlassen, d.h. unterbrochen werden.
- | Nebenläufigkeit und nebenläufige Mengenverarbeitung können definiert werden.
- | Kontrollflüsse, die im Diagramm sehr weite Entfernungen zurücklegen, können mit Hilfe von Konnektoren vereinfacht dargestellt werden.
- | Synchronisationsbalken können individuelle Zusammenführungs- Spezifikationen haben.

4.2 Zusammenfassung

In dieser Ausarbeitung, werden die grundlegende Syntax und Semantik von Aktivitätsdiagrammen in UML2.0 erklärt. Die Syntax hat durch Abstrakt und Konkret eingeführt. Die Semantik ist basiert auf Vollständige Petri-Netze, bzw. Prozedurale Petri-Netze. Aber hier hat keine Erklärung über Datenflüsse, weil sie Petri-Netze auf hoher Ebene als ihre Domäne brauchen. Durch diese Ausarbeitung können wir das einfache Ereignis in Aktivitätsdiagramme darstellen.

Literature:

- [Stö04] Harald Störrle: Semantics of UML 2.0 Activities, Ludwig-Maximilians-Universität, München, Germany.
- [Oes04] Bernd Oestereich: Die UML2.0 Kurzreferenz für die Praxis, Oldenbourg Verlag, München Wien, 2004, ISBN 3-486-27604-2
- [Pen04] Tom Pender: UML Bible, Publishing House of Electronics Industry, Beijing, 2004, ISBN 7-5053-9538-6
- [Sch03/04] Johann Schlichter: Einführung in Informatik III, Technische Universität München, 2003/2004
- [CanXX] C.Canevet, S.Gilmore, J.Hillston, L.Kloul, P.Stevens: Analysing UML2.0 activity diagrams in the software performance engineering process, The University of Edinburgh, Scotland
- [Sup04] UML2.0 Superstructure Specification
- [KanXX] Christian Kanhäuser: Unified Modeling Language (UML), technische Universität Wien,
<http://www.uml.org>
<http://www.oose.de/uml.htm>