2.5 Aktivitätsdiagramme

In diesem Kapitel widmen wir uns einer Diagrammform zur Modellierung von Abläufen. Die Aktivitätsdiagramme der UML 1.x werden verbreitet in den unterschiedlichsten Bereichen eingesetzt: von der Geschäftsprozessmodellierung [Oestereich2004c] über die Systemanforderungsanalyse bis hin zu Algorithmen. Hierbei hat man viel Erfahrung mit dem UML-spezifischen Diagramm gesammelt. Insbesondere wurden Probleme erkannt und Grenzen ausgelotet. Diese Erkenntnisse sind in die Aktivitätsdiagramme der UML 2 eingeflossen.

Das Aktivitätsmodell ist erheblich überarbeitet worden. Während beispielsweise die Aktivitäten der UML 1.x eine spezielle Form der Zustandsautomaten sind, haben die Aktivitäten der UML 2 keine Beziehung mehr zu den Zustandsautomaten. Sie entsprechen viel mehr der Semantik der Petri-Netze und unterstützen wesentlich besser die Ablaufsemantik.

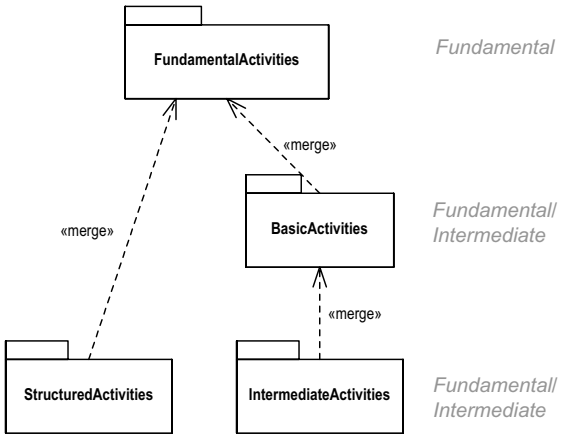


Abb. 2-106 Paketstruktur Aktivitätsdiagramme (*activity diagram*)

2.5.1 Prüfungsthemen

Die Pakete *BasicActivities* und *IntermediateActivities* enthalten Elemente, die auch Bestandteil der Fundamental-Zertifizierung sind.

Folgende Themen werden im Intermediate-Test behandelt:

- Objektknoten (*ObjectNode*)
- Kontrollknoten (*ControlNode*)
- Partition (*ActivityPartition*)
- Strukturknoten (*StructuredActivityNode*)
- Entscheidungsknoten, Schleifenknoten (*ConditionalNode, LoopNode*)
- Ausnahmebehandlung (*ExceptionHandler*)

Prüfungsanteil insgesamt: 15%

2.5.2 Objektknoten (*ObjectNode*)

Pin, Aktivitätsparameterknoten:
Fundamental

Der Objektknoten ist abstrakte Oberklasse für mehrere konkrete Knoten. Der Pin (*Pin*) und der Aktivitätsparameterknoten (*ActivityParameterNode*) sind beispielsweise konkrete Objektknoten, die bereits Bestandteil der Fundamental-Zertifizierung sind. In diesem Kapitel wird ein weiterer konkreter Objektknoten besprochen: der Zentralpuffer (*CentralBufferNode*).

Definition

Ein Zentralpuffer (*CentralBufferNode*) ist ein spezieller Objektknoten, der nicht mit einer Aktion oder einer Aktivität verbunden ist.

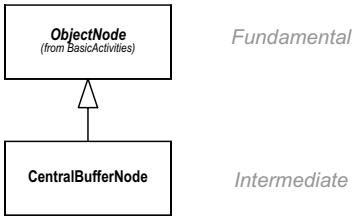


Abb. 2-107 Einordnung Zentralpuffer (*CentralBufferNode*)

Notation und Semantik

☑ 2.5.2.1

Alle Objektknoten (*ObjectNode*) haben eine Pufferfunktionalität. Pins (*Pin*) und Aktivitätsparameterknoten (*ActivityParameterNode*) sind spezielle Objektknoten (*ObjectNode*), die direkt mit einer Aktion bzw. Aktivität verbunden sind. Ein Zentralpuffer (*CentralBufferNode*) ist nur mit anderen Objektknoten oder Kontrollknoten (*ControlNode*) verbunden.

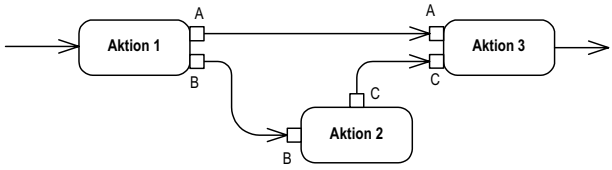
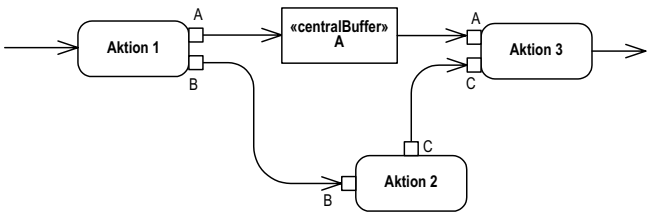


Abb. 2-108 Aktivitätsausschnitt mit Deadlock

☑ 2.5.2.2
Tokenflusssemantik:
Fundamental

In Abb. 2-108 ist ein Ablaufmuster dargestellt, das in der Praxis recht geläufig ist. In der Abbildung ist allerdings ein Modellierungsfehler. Das Objekttoken A, welches von der Aktion 1 kommt, kann nicht weitergeleitet werden, da Aktion 3 nicht bereit ist, das Token aufzunehmen. Das wird erst der Fall sein, wenn Aktion 2 am Ausgabepin das Objekttoken C bereitstellt, was wiederum erst passiert, wenn Aktion 1 Objekttoken B weiterleitet. Damit haben wir einen Deadlock.



Benötigt wird jetzt eine Möglichkeit, das Objekttoken A „zwischenzulagern“, damit Aktion 1 das Token B freigeben kann. Diese Funktionalität wird von dem Zentralpuffer erfüllt (Abb. 2-109).

Abb. 2-109 Aktivitätsausschnitt mit Zentralpuffer

Die ein- und ausgehenden Kanten eines Zentralpuffers haben Oder-Semantik, d.h., das Modell in Abb. 2-110 funktioniert auch, wenn Aktion 4 nicht aktiviert wird.

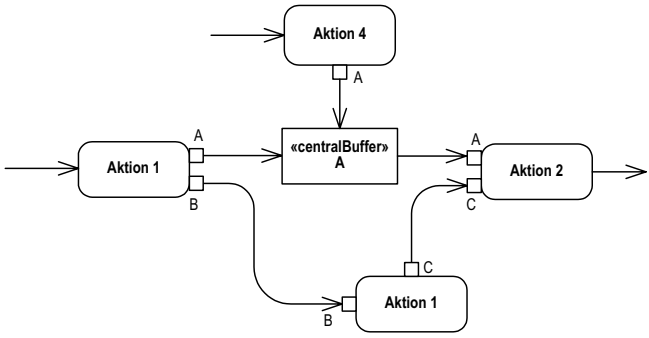


Abb. 2-110 Aktivitätsausschnitt mit Zentralpuffer (Oder-Semantik)

Der Zentralpuffer wird als Rechteck mit Schlüsselwort *centralBuffer* notiert. Unterhalb des Schlüsselwortes steht der Name der Objekttoken, die der Knoten aufnehmen kann. Der Name wird wie bei einem Pin notiert.

Metamodell

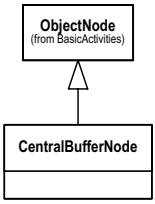


Abb. 2-111 Metamodell Zentralpuffer (*CentralBufferNode*)

Checkliste

Zentralpuffer (*CentralBufferNode*):

☑ 2.5.2.1: Nennen Sie die Eigenschaften eines Zentralpuffers (*CentralBufferNode*).

☑ 2.5.2.2: Erläutern Sie die Tokenflusssemantik beim Zentralpuffer (*CentralBufferNode*).

2.5.3 Kontrollknoten (*ControlNode*)

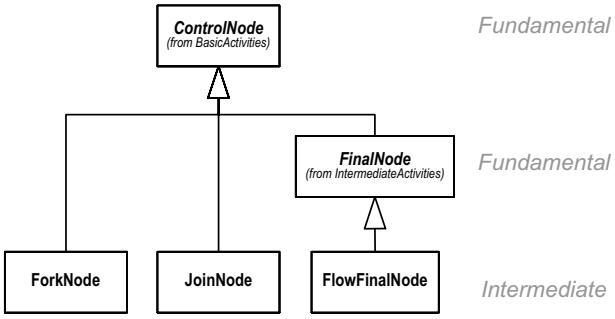
Die Kontrollknoten sind auch Teil der Fundamental-Zertifizierung. Konkret werden dort der Startknoten (*InitialNode*), der Endknoten (*ActivityFinalNode*), die Entscheidung (*DecisionNode*) und die Zusammenführung (*MergeNode*) behandelt. In der Intermediate-Zertifizierung wird diese Liste erweitert um das Splitting (*ForkNode*), die Synchronisation (*JoinNode*) und das Ablaufende (*FlowFinalNode*).

Definition

Das Splitting (*ForkNode*) ist ein spezieller Kontrollknoten, der einen eingehenden Kontroll- oder Objektfluss in mehrere ausgehende Kontroll- oder Objektflüsse aufsplittet.

Die Synchronisation (*JoinNode*) ist ein spezieller Kontrollknoten, der mehrere eingehende Kontroll- oder Objektflüsse synchronisiert und genau einen ausgehenden Kontroll- oder Objektfluss hat.

Das Ablaufende (*FlowFinalNode*) ist ein spezieller Kontrollknoten, der



einen Ablauf beendet. Jedes eintreffende Token wird zerstört.

Abb. 2-112 Einordnung Splitting (*ForkNode*), Synchronisation (*JoinNode*) und Ablaufende (*FlowFinalNode*)

Notation und Semantik

Die Abb. 2-113 zeigt ein Splitting (*ForkNode*), das einen eingehenden Kontrollfluss in zwei ausgehende Kontrollflüsse aufsplittet. Genauer gesagt wird das eingehende Kontrolltoken kopiert und an jeder ausgehenden Kante wird ein Kontrolltoken zur Verfügung gestellt. ☑ 2.5.3.1

Das Splitting wird als schwarzer Balken mit einer eingehenden und mehreren ausgehenden Kanten notiert.

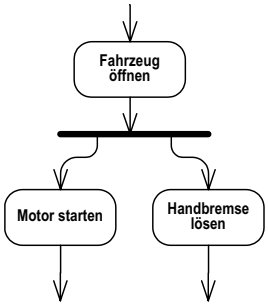


Abb. 2-113 Beispiel Splitting (*ForkNode*)

☑ 2.5.3.2

Die Abb. 2-114 zeigt ein Splitting mit Objektfluss. Nach der allgemeinen Ablaufsemantik gilt, dass ein Token erst weitergeleitet wird, wenn alle Kanten und Zielknoten bereit sind. Zusätzlich gilt, dass ein Token nicht an einem Kontrollknoten warten darf. Diese Regeln führen zu einem Konflikt mit der Semantik eines Splittings, dass die ausgehenden Abläufe unabhängig voneinander sind. In Abb. 2-114 kann die Bedingung *[Mietwagen notwendig]* das gesamte Splitting blockieren.

Daher gibt es zwei Ausnahmen zur allgemeinen Ablaufsemantik:

1. Damit eine Bedingung (*Guard*), die zu *false* ausgewertet wird, nicht das gesamte Splitting blockiert, wird an der betroffenen Kante kein Token zur Verfügung gestellt Alle anderen Kanten werden aber bedient. In Abb. 2-114 wird in diesem Fall kein Mietwagen, aber Hotel und Flug gebucht.
2. Normalerweise könnte das Splitting Objekttoken auch nicht weiterleiten, wenn ein Zielknoten nicht bereit ist, das Objekttoken aufzunehmen. In diesem Fall wartet das Objekttoken der betroffenen Kante am Splitting. Das ist eine Ausnahme von der allgemeinen Regel, dass ein Token nicht an einem Kontrolknoten warten darf. Alle anderen Kanten werden bedient. Das wartende Objekttoken geht nicht verloren, sondern wird nach FIFO-Semantik an den Zielknoten weitergeleitet, sobald dieser bereit ist.

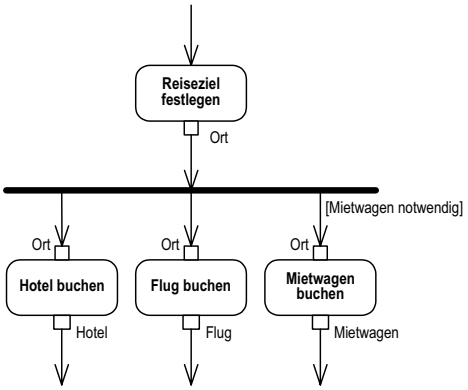


Abb. 2-114 Beispiel Splitting (*ForkNode*) mit Objektfluss

Die Synchronisation (*JoinNode*) hat mehrere eingehende Kanten und eine ausgehende Kante. Im Gegensatz zur Zusammenführung (*MergeNode*), werden die Abläufe basierend auf drei Regeln synchronisiert:

☑ 2.5.3.3 Zusammenführung: Fundamental ☑ 2.5.3.4

1. An allen eingehenden Kanten müssen Token zur Verfügung stehen, damit die Synchronisation an der ausgehenden Kante Token bereitstellt.
2. Wenn alle eingehenden Token Kontrolltoken sind, wird an der ausgehenden Kante genau ein Kontrolltoken zur Verfügung gestellt.
3. Wenn an den eingehenden Kanten Objekt- und Kontrolltoken eintreffen, werden nur die Objekttoken weitergeleitet. Die Weiterleitung geschieht in derselben Reihenfolge, in der sie an der Synchronisation eintreffen.

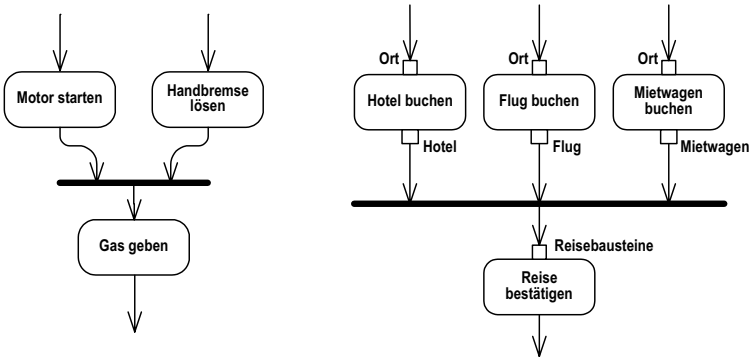


Abb. 2-115 Aktivitätsausschnitt mit Synchronisation (*JoinNode*)

Die Synchronisation wird als schwarzer Balken mit mehreren eingehenden und einer ausgehenden Kante notiert.

Es ist zulässig, im Diagramm direkt aufeinander folgendes Splitting und Synchronisation (bzw. umgekehrt) nur als einen schwarzen Balken mit mehreren eingehenden und ausgehenden Kanten zu zeichnen. Im Modell sind es aber weiterhin zwei Kontrollknoten.

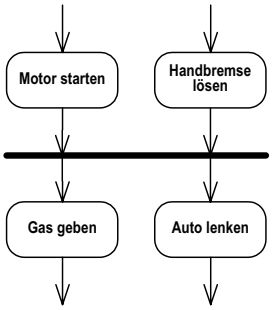


Abb. 2-116 Splitting und Synchronisation mit einem Symbol

☑ 2.5.3.5

Das Ablaufende (*FlowFinalNode*) beendet einen einzelnen Ablauf. Anders als der Endknoten (*ActivityFinalNode*), der beim Eintreffen eines Tokens die gesamte Aktivität beendet, wird beim Ablaufende nur das eintreffende Token zerstört. Alle anderen Token in der Aktivität werden dadurch nicht beeinflusst.

Wenn das letzte Token einer Aktivität ein Ablaufende erreicht, wird die gesamte Aktivität beendet.

Das Ablaufende wird als Kreis mit Kreuz notiert.

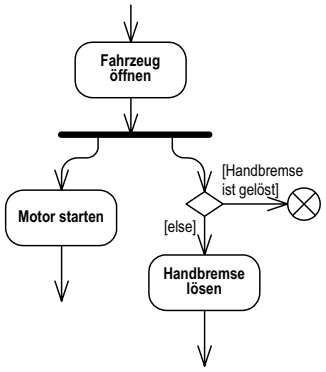


Abb. 2-117 Aktivitätsausschnitt mit Ablaufende (*FlowFinalNode*)

Metamodell

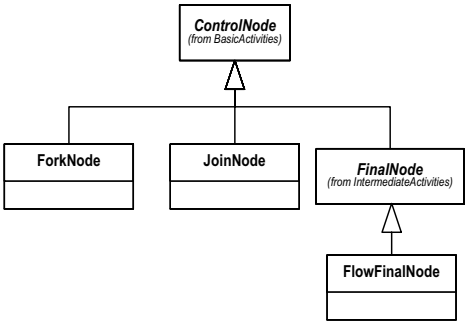


Abb. 2-118 Metamodell Kontrollknoten (*ControlNode*)

Checkliste

Kontrollknoten (*ControlNode*):

- ☑ 2.5.3.1: Nennen Sie die Eigenschaften eines Splittings (*ForkNode*). Wie viele Token fließen nach dem Splitting (*ForkNode*)?
- ☑ 2.5.3.2: Erläutern Sie die Tokenflusssemantik beim Splitting (*ForkNode*).
- ☑ 2.5.3.3: Nennen Sie die Eigenschaften einer Synchronisation (*JoinNode*).
- ☑ 2.5.3.4: Erläutern Sie die Tokenflusssemantik bei der Synchronisation (*JoinNode*). Wie viele Token fließen nach einer Synchronisation (*JoinNode*)?
- ☑ 2.5.3.5: Nennen Sie die Eigenschaften eines Ablaufendes (*FlowFinalNode*).

2.5.4 Partition (*ActivityPartition*)

Definition

Die Aktivitätsgruppe (*ActivityGroup*) ist ein abstraktes Element, das eine Menge von Knoten und Kanten einer Aktivität gruppiert.

Die Partition (*ActivityPartition*) ist eine konkrete Aktivitätsgruppe.

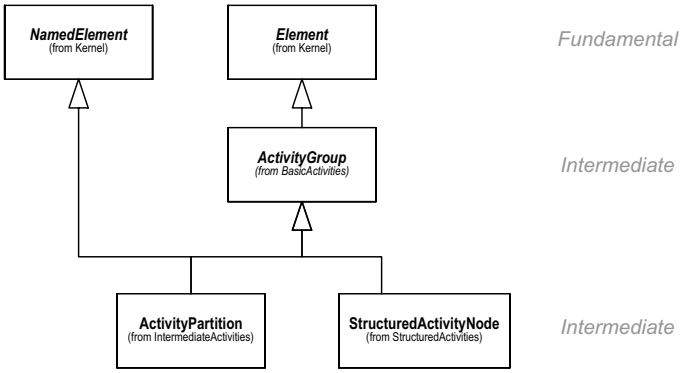


Abb. 2-119 Überblick Partition

Notation und Semantik

☑ 2.5.4.1 Die Aktivitätsgruppe (*ActivityGroup*) ist abstrakt und kann somit nicht direkt verwendet werden. Es gibt keine Notation.

Aktivitätsgruppen dürfen verschachtelt werden, d.h., eine Aktivitätsgruppe kann Aktivitätsgruppen enthalten.

Die Knoten und Kanten in der Aktivitätsgruppe müssen alle derselben Aktivität gehören. Sie dürfen Teil mehrerer Aktivitätsgruppen sein, wenn die Aktivitätsgruppen keine Ober- bzw. Untergruppen zueinander sind.

☑ 2.5.4.2 Eine Partition (*ActivityPartition*) ist eine Spezialisierung von Aktivitätsgruppe und benennbarem Element (*NamedElement*). Letzteres ist wichtig, um der Partition einen Namen geben zu können. Eine Eigenschaft, die die Aktivitätsgruppe nicht hat.

Die Partition wird als Rechteck mit einem Namensbereich und einem Bereich für die enthaltenen Knoten und Kanten notiert. Die meist vertikale Ausrichtung ist nicht zwingend erforderlich.

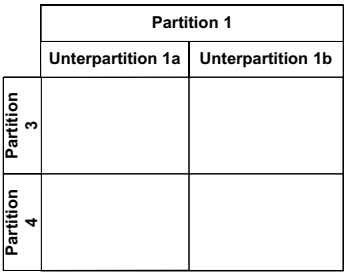


Abb. 2-120 Notation Partition (*ActivityPartition*)

Eine Partition kann Unterpartitionen besitzen. Damit ist es möglich, hierarchische Strukturen zu modellieren. Anders als bei der Aktivitätsgruppe dürfen sich Knoten und Kanten nicht in mehreren Partitionen gleichzeitig befinden. Eine Ausnahme ist die Dimensionseigenschaft (siehe unten).

Das gemeinsame Kriterium zur Bildung der Partitionen kann explizit modelliert werden (Assoziationsende *represents*). Assoziiert wird die abstrakte Basisklasse Element. Somit kann also jedes Modellelement der UML Gruppierungskriterium einer Partition sein. M1

Im Diagramm kann das Element in einem eigenen Bereich oberhalb des Partitionsnamens notiert werden. Zusätzlich wird als Schlüsselwort der Name der zugehörigen Metaklasse angegeben.

Die Eigenschaft, außerhalb des Fokus der Partitionierung zu liegen, ist ein besonderes Gruppierungskriterium. Die entsprechende Partition wird mit *«external»* gekennzeichnet. Das kann beispielsweise eingesetzt werden, um mit einer Aktivität den Ablauf eines Anwendungsfalles zu modellieren. Die mit *«external»* gekennzeichnete Partition enthält dann die Aktionen der Akteure.

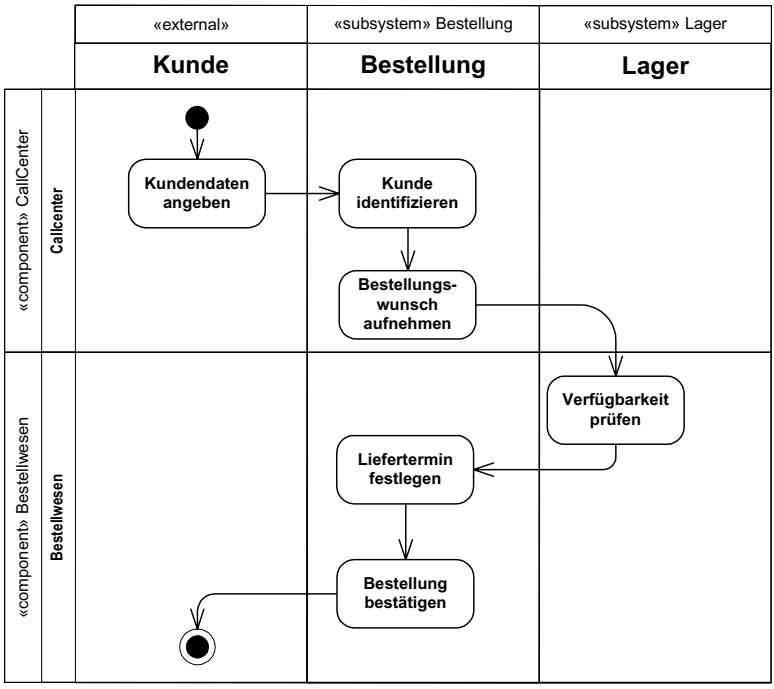


Abb. 2-121 Beispiel Partition

Eine Partition kann eine Dimension haben. Solch eine Partition darf keine Unterpartition sein. Sie steht orthogonal auf den obersten Partitionen, die keine Dimension besitzen. In Abb. 2-121 haben die Partitionen, die nach Komponenten gruppiert sind, eine Dimension.

Knoten und Kanten dürfen als Ausnahme zur allgemeinen Regel in mehreren Partitionen enthalten sein, wenn sie in unterschiedlichen Dimensionen liegen.

Die Bahnen einer Partition sind eine erhebliche Einschränkung der Freiheit im Layout eines Aktivitätsdiagramms. Es ist daher auch eine alternative Notation möglich, die diese Restriktion nicht mit sich bringt. Der Name der Partition wird in runden Klammern oberhalb des Aktionsnamens notiert. Verschachtelte Partitionen sind mit zwei Doppelpunkten getrennt, Dimensionen werden mit Komma getrennt (Abb. 2-122).

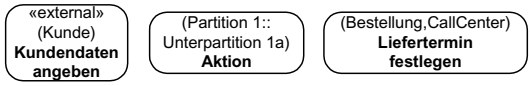


Abb. 2-122 Alternative Notation einer Partition

Metamodell

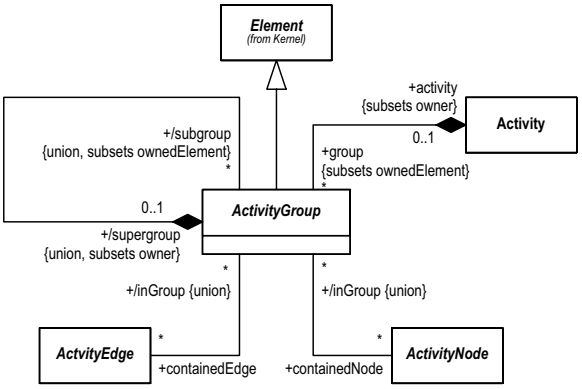


Abb. 2-123 Metamodell Aktivitätsgruppe (*ActivityPartition*)

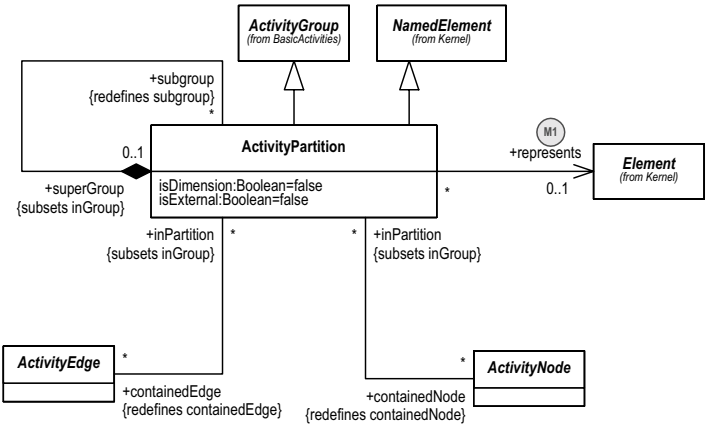


Abb. 2-124 Metamodell Partition (*ActivityPartition*)

Checkliste

Aktivitätsgruppe (*ActivityGroup*):

☑ 2.5.4.1: Nennen Sie die Eigenschaften einer Aktivitätsgruppe (*ActivityGroup*).

☑ 2.5.4.2: Nennen Sie die Eigenschaften einer Partition (*ActivityPartition*). Worauf bezieht sich eine Partition (*ActivityPartition*)?

2.5.5 Strukturknoten (*StructuredActivityNode*)

Definition

Der Strukturknoten (*StructuredActivityNode*) ist ein ausführbarer Knoten (*ExecutableNode*) und eine Aktivitätsgruppe (*ActivityGroup*), d.h., er enthält weitere Knoten und Kanten. Die enthaltenen Elemente dürfen in keinem anderen Strukturknoten vorkommen.

Die Variable (*Variable*) gehört zu einem Strukturknoten und enthält Werte, auf die alle Aktionen des Strukturknotens zugreifen können.

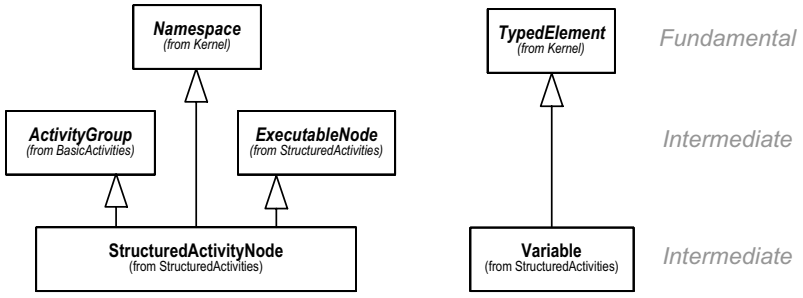


Abb. 2-125 Einordnung Strukturknoten (*StructuredActivityNode*) und Variable (*Variable*)

Notation und Semantik

☑ 2.5.5.1

Im Gegensatz zur einzelnen Aktion, die ein atomarer ausführbarer Knoten ist, stellt der Strukturknoten (*StructuredActivityNode*) eine nicht atomare ausführbare Gruppe von Aktionen dar.

Der Strukturknoten ist vergleichbar mit einer Operation in einer Programmiersprache, die einzelne Aktion entspricht dann einer einzelnen Anweisung in der Operation. Ebenso wie Programmiersprachen-Operationen lokale Variablen besitzen können, lassen sich auch innerhalb eines Strukturknotens Variablen definieren (siehe unten).

Start- und Endknoten: Fundamental

Ein Strukturknoten kann Start- und Endknoten enthalten (*InitialNode*, *ActivityFinalNode*). Im Kontext eines Strukturknotens beginnt der Kontrollfluss an einem Startknoten erst, wenn der Strukturknoten ausgeführt wird. Der Endknoten beendet im Kontext eines Strukturknotens nicht die gesamte Aktivität, sondern nur den Strukturknoten.

Die Angabe von Start- und Endknoten ist nicht zwingend erforderlich. Wenn der Strukturknoten aufgerufen wird, startet der Ablauf innerhalb des Strukturknotens an allen Knoten, die keine eingehenden Kanten haben. Umgekehrt gilt dann Entsprechendes für den Endknoten.

Innerhalb eines Strukturknotens wird kein Ablauf gestartet, bevor nicht der Strukturknoten ausgeführt wird.

Eine Variable (*Variable*) ist ein typisierbares Element (*TypedElement*), auf das alle Aktionen des zugehörigen Strukturknotens zugreifen können. Ein expliziter Datenaustausch über Objektfluss ist dann nicht erforderlich. Die Variable gehört zu einem Strukturknoten.

M1
☑ 2.5.5.2
Typisierbares Element:: Fundamental

Der Strukturknoten wird als gestricheltes Rechteck mit abgerundeten Ecken notiert. Im oberen Bereich steht das Schlüsselwort *structured* und der Name des Strukturknotens. Im übrigen Bereich werden die enthaltenen Knoten und Kanten notiert.

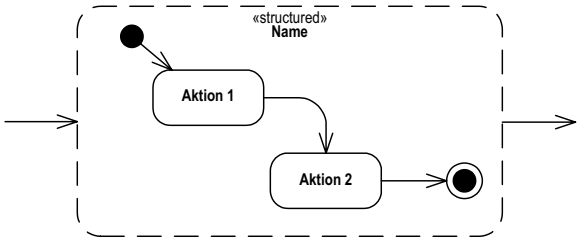


Abb. 2-126 Notation Strukturknoten (*StructuredActivityNode*)

Metamodell

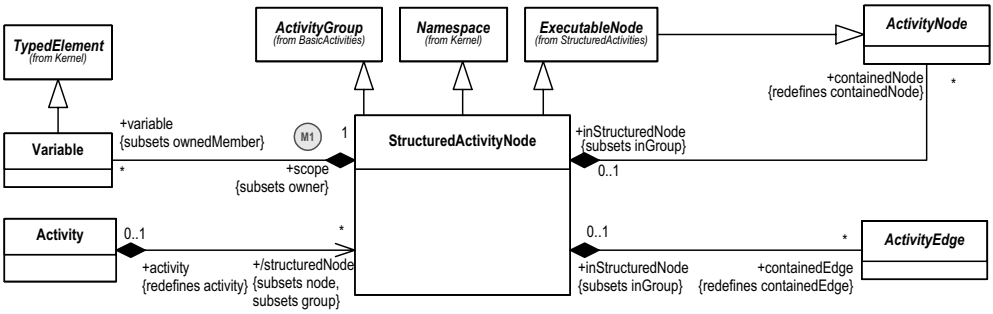


Abb. 2-127 Metamodell Strukturknoten (*StructuredActivityNode*)

Checkliste

Strukturknoten (*StructuredActivityNode*):

☑ 2.5.5.1: Nennen Sie die Eigenschaften des Strukturknoten (*StructuredActivityNode*).

☑ 2.5.5.2: Nennen Sie die Eigenschaften der Variablen (*Variable*). Zu wem gehört die Variable (*Variable*)?

2.5.6 Entscheidungsknoten, Schleifenknoten, Sequenzknoten

Verzweigungen, Schleifen und Sequenzen sind weit verbreitete Muster in Abläufen. Der Entscheidungs-, der Schleifen- und der Sequenzknoten bieten hierfür eine Struktur, um diese Muster kompakt und übersichtlich in einer Aktivität auszudrücken.

Definition

Der Entscheidungsknoten (*ConditionalNode*) ist ein spezieller Strukturknoten (*StructuredActivityNode*), der der *if-then-else*-Semantik entspricht.

Die bedingte Verzweigung (*Clause*) besteht aus einem Test und einem Rumpf *(Body)*, der ausgeführt wird, wenn der Test *true* ergibt.

Der Schleifenknoten (*LoopNode*) ist ein spezieller Strukturknoten (*StructuredActivityNode*), der die *while*- bzw. *do-while*-Semantik abbildet.

Der Sequenzknoten (*SequenceNode*) führt ausführbare Knoten (*ExecutableNode*) der Reihenfolge nach aus.

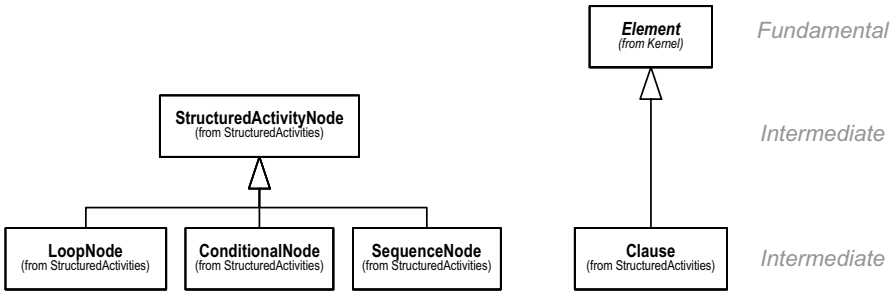


Abb. 2-128 Einordnung Entscheidungsknoten (*ConditionalNode*), bedingte Verzweigung (*Clause*), Schleifenknoten (*LoopNode*) und Sequenzknoten (*SequenceNode*)

Notation und Semantik

☑ 2.5.6.1 Die bedingte Verzweigung (*Clause*) gruppiert eine Menge von Knoten, die in zwei Kategorien unterteilt sind: die Test- und die Rumpfknoten.

Die Testknoten führen gemeinsam einen Test mit booleschem Ergebnis aus. Einer der Testknoten muss einen Ausgabepin besitzen, der das Testergebnis aufnimmt (*Decider*). Wenn das Testergebnis *true* ist, werden die Rumpfknoten ausgeführt. Der Ablauf beginnt an all den Knoten, die keine eingehenden Kanten haben (*front end nodes*), und endet in den Knoten, die keine ausgehenden Kanten haben (*back end nodes*).

Der Entscheidungsknoten (*ConditionalNode*) besitzt eine Menge von Verzweigungen. Wenn die Laufzeitumgebung des Aktivitätsmodells es erlaubt, können alle Tests der Verzweigungen parallel ausgeführt werden. Der Ablauf ist dann nicht deterministisch, wenn mehrere Tests das Ergebnis *true* ergeben. Um das zu verhindern, können die bedingten Verzweigungen geordnet werden (Assoziationsenden *predecessorClause* und *successorClause*). ☑ 2.5.6.2 M1

Eine besondere bedingte Verzweigung ist die *else*-Verzweigung. Sie ist Nachfolger aller anderen Verzweigungen eines Entscheidungsknotens und ihr Test ergibt immer *true*.

Die Eigenschaft *isAssured=true* sichert zu, dass mindestens ein Test mit *true* ausgewertet wird. Die Eigenschaft *isDeterminate=true* sichert zu, dass bei nebenläufigen Auswertungen der Testbereiche nur maximal ein Rumpf zur Ausführung kommt. M2

Der Entscheidungsknoten wird als gestricheltes Rechteck mit abgerundeten Ecken notiert. Die Notation wird vom Strukturknoten geerbt.

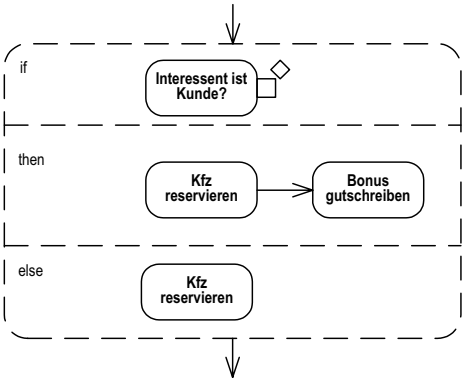


Abb. 2-129 Notation und Beispiel Entscheidungsknoten (*ConditionalNode*)

Der Schleifenknoten (*LoopNode*) ist wie der Entscheidungsknoten (*ConditionalNode*) ein spezieller Strukturknoten (*StructuredActivityNode*). Entsprechend der Semantik einer Schleife setzt der Knoten sich aus folgenden drei Bereichen zusammen: Initialisierung, Test und Rumpf. ☑ 2.5.6.3

Alle drei Bereiche werden jeweils durch eine Menge von Knoten repräsentiert. Ein Knoten im Testbereich muss einen Ausgabepin besitzen, der das boolesche Testergebnis aufnimmt (*Decider*).

☑ 2.5.6.4

M3

Bei der Ausführung eines Schleifenknotens wird zunächst der Initialisierungsbereich ausgeführt. Wenn die Eigenschaft *isTestedFirst* auf *true* gesetzt ist (*while-do*-Semantik), wird als Nächstes der Test durchgeführt. Wenn das Testergebnis *true* ist, wird der Rumpf ausgeführt. Danach wieder der Test usw., bis der Test *false* ergibt.

Die Ausführungsreihenfolge Test – Rumpf wird umgekehrt, wenn die Eigenschaft *isTestedFirst* auf *false* gesetzt ist (*do-while*-Semantik).

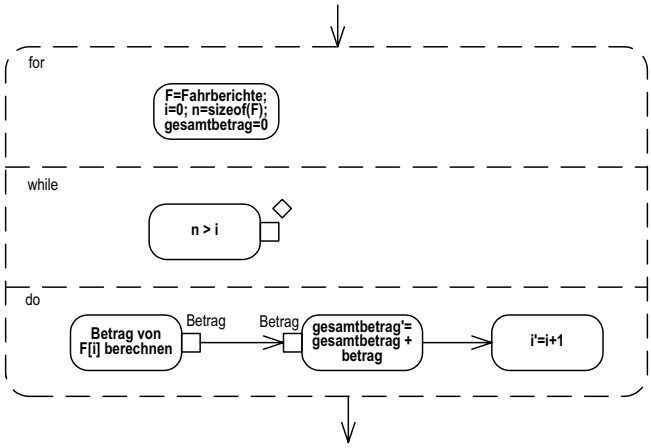


Abb. 2-130 Notation und Beispiel Schleifenknoten (*LoopNode*)

☑ 2.5.6.5

Der Sequenzknoten (*SequenceNode*) verweist auf eine geordnete Liste von ausführbaren Knoten (*ExecutableNode*). Wenn der Sequenzknoten aktiviert wird, werden die ausführbaren Knoten gemäß der Ordnung der Reihe nach ausgeführt. Wenn die Knoten mit Kontroll- oder Objektflüssen verbunden sind, müssen die damit verknüpften Bedingungen erfüllt sein, damit ein Knoten ausgeführt werden kann.

Metamodell

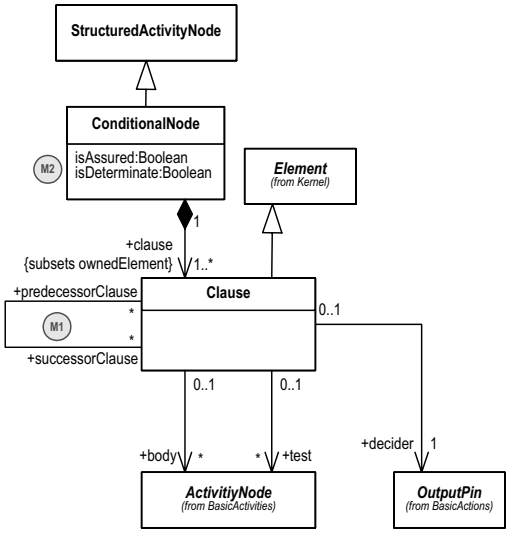


Abb. 2-131 Metamodell Entscheidungsknoten (*ConditionalNode*)

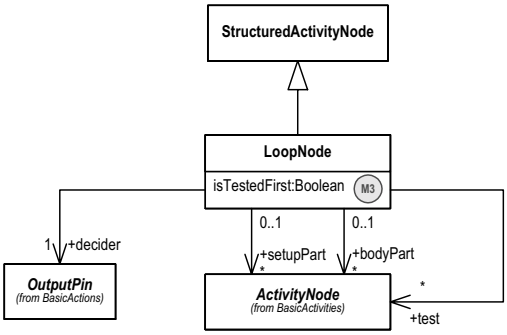


Abb. 2-132 Metamodell Schleifenknoten (*LoopNode*)

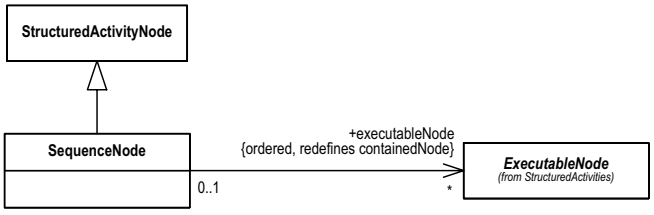


Abb. 2-133 Metamodell Sequenzknoten (*SequenceNode*)

Checkliste

Entscheidungsknoten (*ConditionalNode*), Schleifenknoten (*LoopNode*), Sequenzknoten (*SequenceNode*):

- ☑ 2.5.6.1: Nennen Sie die Eigenschaften der bedingten Verzweigung (*Clause*).
- ☑ 2.5.6.2: Nennen Sie die Eigenschaften des Entscheidungsknotens (*ConditionalNode*).
- ☑ 2.5.6.3: Nennen Sie die Eigenschaften des Schleifenknotens (*LoopNode*).
- ☑ 2.5.6.4: Wie wird beim Schleifenknoten (*LoopNode*) zwischen der *do-while-* und der *while-do*-Semantik unterschieden?
- ☑ 2.5.6.5: Nennen Sie die Eigenschaften des Sequenzknotens (*SequenceNode*).

2.5.7 Ausnahmebehandlung (*ExceptionHandler*)

Definition

Die Ausnahmebehandlung (*ExceptionHandler*) spezifiziert einen ausführbaren Knoten *(ExecutableNode)*, der aufgerufen wird, wenn in einem geschützten ausführbaren Knoten *(ExecutableNode)* eine Ausnahme (*Exception*) auftritt.

Der ausführbare Knoten (*ExecutableNode*) ist abstrakte Basisklasse für Knoten, die ausgeführt werden können. Der Knoten kann von einer Ausnahmebehandlung geschützt werden.

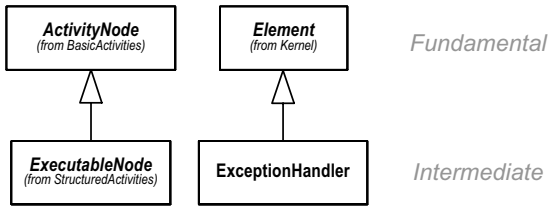


Abb. 2-134 Einordnung Ausnahmebehandlung (*ExceptionHandler*) und ausführbarer Knoten (*ExecutableNode*)

Notation und Semantik

Der ausführbare Knoten (*ExecutableNode*) ist abstrakte Basisklasse für Knoten, die ausgeführt werden können.

Ein ausführbarer Knoten kann mehrere Ausnahmebehandlungen (*ExceptionHandler*) besitzen. Die Ausnahmebehandlung spezifiziert einen oder mehrere Ausnahmetypen, die von ihm „gefangen“ und behandelt werden können. ☑ 2.5.7.1

Das „geworfene“ Ausnahmeobjekt ist Eingangsinformation des ausführbaren Knotens, der die Ausnahmebehandlung durchführt (Assoziationsende *handlerBody*). Der Knoten hat neben der Kante, über die das Ausnahmeobjekt kommt, keine weiteren eingehenden und keine ausgehenden Kanten. Die Rückgabewerte müssen den Rückgabewerten des geschützten Knotens entsprechen. M1

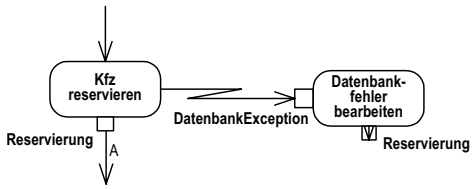


Abb. 2-135 Notation und Beispiel Ausnahmebehandlung (*ExceptionHandler*)

☑ 2.5.7.2

Der Ablauf einer Ausnahmebehandlung ist in Abb. 2-135 dargestellt: Der geschützte Knoten *Kfz reservieren* wird ausgeführt, eine Ausnahme tritt auf und die Ausnahmebehandlung *Datenbankfehler bearbeiten* wird gestartet. Nach Ausführung der Ausnahmebehandlung geht der Ablauf entlang der Kante *A* mit einem Objekttoken *Reservierung* weiter. Die Reservierung ist in diesem Fall Rückgabewert der Ausnahmebehandlung.

Eine Ausnahmebehandlung wird als ausführbarer Knoten mit einem blitzförmigen eingehenden Objektfluss und Eingabepin für das Ausnahmeobjekt notiert.

Metamodell

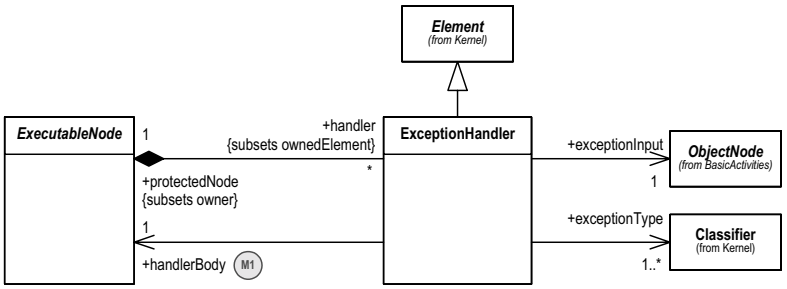


Abb. 2-136 Metamodell Ausnahmebehandlung (*ExceptionHandler*)

Checkliste

Ausnahmebehandlung (*ExceptionHandler*):

☑ 2.5.7.1: Nennen Sie die Eigenschaften einer Ausnahmebehandlung (*ExceptionHandler*). Was wird durch eine Ausnahmebehandlung geschützt?

☑ 2.5.7.2: Erläutern Sie den Ablauf einer Ausnahmebehandlung (*ExceptionHandler*).