4.2 Aktivitätsdiagramm

4.2.1 Einführung

Fokus auf prozeduraler Verarbeitung in Form von Kontroll- und Datenfluss zwischen Aktionen

Der Schwerpunkt von Aktivitätsdiagrammen liegt auf prozeduralen Verarbeitungsaspekten. Ein Aktivitätsdiagramm spezifiziert den Kontroll- und Datenfluss zwischen verschiedenen Arbeitsschritten, den Aktionen, die zur Realisierung einer Aktivität notwendig sind.

In UML1 Vermischung von Konzepten aus Zustandsdiagrammen und ablauforientierter Notation

In UML1 stellen Aktivitätsdiagramme eine spezielle Form von Zustandsdiagrammen dar, deren Notation jedoch einen ablauforientierten Charakter aufweist, wodurch die Ausdrucksmächtigkeit eingeschränkt ist und der Einsatz nicht immer klar erscheint.

Neue konzeptuelle Basis bei gleichzeitigem Festhalten an ablauforientierter Notation

Dieser Kritik Rechnung tragend, wurden Aktivitätsdiagramme in UML2 auf eine völlig neue konzeptionelle Basis gestellt. Aktivitätsdiagramme basieren nun nicht mehr auf Konzepten des Zustandsdiagramms, sondern verwenden ablauforientierte Sprachkonzepte, die ihre Wurzeln in aktuellen Sprachen zur Definition Web-Service-basierter Geschäftsprozesse [BPEL03], aber auch in altbewährten Konzepten zur Beschreibung nebenläufiger, kommunizierender Prozesse haben, wie dem *Tokenkonzept* aus Petrinetzen [Petr62]. Im Hinblick auf die Notation wurde jedoch darauf geachtet, soweit möglich an bestehenden Notationselementen festzuhalten, wenngleich die Bedeutung zum Teil neu definiert wurde.

Modellierung objektorientierter und nicht-objektorien-tierter Systeme gleicher-massen unterstützt

Ein besonderes Merkmal von Aktivitätsdiagrammen liegt darin, dass neben der Modellierung objektorientierter Systeme inbesondere auch die Modellierung nicht-objektorientierter Systeme unterstützt wird. Aktivitäten können unabhängig von Objekten definiert werden, wodurch beispielsweise auch Funktionsbibliotheken oder Geschäftsprozesse modelliert werden können.

Sprachkonzepte und Notationsvarianten decken ein breites Anwendungsgebiet ab

Um diesen breiten Einsatzbereich abzudecken, stellen Aktivitätsdiagramme zum Teil alternative Sprachkonzepte und Notationselemente zur Verfügung, die in dem einen oder anderen Anwendungsbereich besonders nützlich sind, jedoch auch miteinander kombiniert werden können. Beispielsweise ist zur Modellierung einer Fallunterscheidung ein *Entscheidungsknoten*, der eine ablaufdiagramm-ähnliche Notation aufweist, eher für Geschäftsprozessmodellierer zweckmäßig, während ein *Konditionalknoten*, für den eine struktogramm-ähnliche Notation verwendet werden kann, eher den »Gewohnheiten« eines Softwareentwicklers entgegenkommt.

Neben vorgeschlagener grafischer Notation beliebige andere Formen – z.B. Pseudo-code erlaubt

Prinzipiell ist der UML-Standard offen im Hinblick auf die, für die Repräsentation von Aktivitäten, verwendete Notation und erlaubt neben den vorgeschlagenen ablauforientierten Notationselementen der Aktivitätsdiagramme auch beliebige andere Notationsformen, wie z.B. Struktogramme oder einfach nur Pseudocode.

»Workflow Patterns« als Basis komplexer Geschäftsprozesse

Im Hinblick auf die Verwendung von Aktivitätsdiagrammen insbesondere für den Bereich der Geschäftsprozessmodellierung haben sich neben ablauforientierten Notationselementen auch eine Reihe von wiederkehrenden Kontroll- und Datenflusskonstrukten - sogenannte »Workflow Patterns« herausgebildet, deren Verwendung sich besonders bei komplexen Abläufen als sehr nützlich erwiesen hat. Für eine Übersicht derartiger Muster und deren Modellierung auf Basis der Konzepte von UML2-Aktivitätsdiagrammen sei auf Wohed et al. verwiesen [Wohe04], ein Leitfaden zur Geschäftsprozessmodellierung auf Basis von UML2 wird in Oestereich et al. gegeben [Oest03].

4.2.2 Eigenschaften einer Aktivität

Eine Aktivität spezifiziert benutzerdefiniertes Verhalten auf unterschiedlichen Granularitätsebenen

Ein Aktivitätsdiagramm erlaubt die Spezifikation von *benutzerdefiniertem benanntem Verhalten* in Form einer Aktivität. Eine Aktivität kann auf einer fein-granularen Ebene beispielsweise die Methode einer Operation in Form von einzelnen Anweisungen definieren, oder aber auf einer grob-granularen Ebene beispielsweise einen Geschäftsprozess oder den Ablauf eines Anwendungsfalls spezifizieren.

Eine Aktivität in UML1 entspricht einer atomaren Aktion in UML2

Eine Aktivität in UML2 umfasst Aktionen, Kontroll- und Datenflüsse

Eine Aktivität in UML2 unterscheidet sich damit grundlegend von dem in UML1 verwendeten Aktivitätsbegriff. In UML1 ist eine Aktivität atomarer Bestandteil eines Aktivitätsdiagramms, während in UML2 eine Aktivität die gesamte Verhaltensbeschreibung in einem Aktivitätsdiagramm umfassen kann. Atomare Bestandteile eines Aktivitätsdiagramms sind in UML2 *Aktionen*. Vordefinierte Aktionen erlauben es, Operationen auf Objekte auszuführen oder anderes Verhalten aufzurufen, wodurch die Wiederverwendung anderer Aktivitäten und Verhaltensbeschreibungen unterstützt wird (siehe Kapitel 4.2.13).

Aktionen leisten die Arbeit, Kontroll- und Datenflüsse legen potentielle »Abläufe« fest

Aktionen leisten damit »die eigentliche Arbeit« für eine Aktivität, während entsprechende Kontroll- und Datenflusskonzepte eine Menge potentieller *»Abläufe«* einer Aktivität zur Laufzeit festlegen, d.h. *wann* eine Aktion ausgeführt werden kann und *welche Daten* dazu erforderlich sind (vgl. dazu den Begriff des »Trace« in Kapitel 4.4.3, »*Interaktion als Abfolge von Ereignisspezifikationen*«, S. 269). Von diesen potentiellen Abläufen kann zur Laufzeit entweder ein bestimmter auftreten oder aber es werden aufgrund von Nebenläufigkeit oder einem Mehrfachaufruf der Aktivität mehrere Abläufe ausgeführt.

Aktivitätszuordnung zu Classifier indirekt über eine Methode oder direkt

Im Sinne des objektorientierten Prinzips der Einheit von Struktur und Verhalten kann eine Aktivität einem *Classifier* zugeordnet werden und damit auf die Werte des *Classifier*-Objekts zugreifen. Die Aktivität fungiert dabei entweder *als Methode* des *Classifiers* oder ist diesem *direkt zugeordnet* und spezifiziert damit das Verhalten des *Classifiers* selbst.

Aktivitätsausführung durch Operationsaufruf oder Classifier-Instanzierung

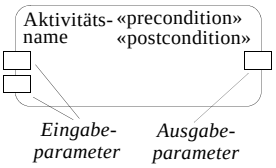
Löschen des Classifier-Objekts

Fungiert die Aktivität als Methode des *Classifiers*, so wird sie ausgeführt, sobald die entsprechende Operation aufgerufen wird, die durch die Methode und damit durch die Aktivität realisiert wird. Ist die Aktivität dem *Classifier* direkt zugeordnet, beginnt die Aktivitätsausführung, sobald der *Classifier* instanziert wird. In diesem Fall wird das *Classifier*-Objekt am Ende einer Aktivitätsausführung gelöscht. Wird umgekehrt, vor Ende der Aktivitätsausführung, das *Classifier*-Objekt gelöscht, so wird auch die Ausführung der Aktivität beendet.

»Autonome« Aktivität ohne Zuordnung zu einem Classifier

Um nicht nur das objektorientierte Paradigma zu unterstützen, kann eine Aktivität auch »autonom«, d.h. ohne Zuordnung zu einem *Classifier*, definiert werden. Für den Aufruf einer autonomen Aktivität muss eine spezielle vordefinierte Aktion *CallBehaviorAction* verwendet werden (siehe Kapitel 4.2.13).

Aktivität kann Parameter (activity parameter nodes), sowie Vor- und Nachbedingungen aufweisen und definiert einen Namensraum

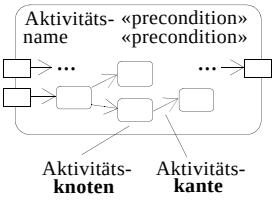


Eine Aktivität wird wie eine Aktion in Form eines abgerundeten Rechtecks dargestellt, wobei deren Name in der linken oberen Ecke angegeben wird. Alternativ dazu kann statt des abgerundeten Rechtecks, die bei allen UML-Diagrammarten verwendbare Rahmennotation verwendet werden, wobei das kleine Pentagon Diagrammart (*ad*) und Name der Aktivität anzeigt (siehe Kapitel 2.4.5, »*Offene Grenzen zwischen verschiedenen Diagrammarten*«, S. 43).

Eine Aktivität kann, wie zum Beispiel bei Operationsaufrufen nützlich, *parametrisiert* werden. Formale Parameter werden durch Rechtecke dargestellt, die die Ränder der Aktivität überlappen, wobei, zur besseren Lesbarkeit, Eingabeparameter am linken oder oberen Rand und Ausgabeparameter am rechten oder unteren Rand, dargestellt werden sollten (siehe Kapitel 4.2.8, Arten von Objektknoten). Diese Konvention ermöglicht ein intuitives Lesen des Diagramms von links oben nach rechts unten. Auch die Angabe von *Vor- und Nachbedingungen* ist möglich, die zu Beginn der Ausführung einer Aktivität bzw. bei deren Beendigung gelten müssen. Die Einschränkungen werden dabei jeweils neben den Schlüsselwörtern *«precondition»* und *«postcondition»* vermerkt. Schließlich stellt eine Aktivität einen *Namensraum* dar, der die Sichtbarkeit von Variablen auf die Aktivität beschränkt.

4.2.3 Bestandteile einer Aktivität

Aktivität ist ein gerichteter Graph bestehend aus Knoten und Kanten



Der Inhalt einer Aktivität ist – analog zu Petrinetzen – ein gerichteter Graph bestehend aus *Aktivitätsknoten* und *Aktivitätskanten*, in weiterer Folge kurz als *Knoten* und *Kanten* bezeichnet. Dabei repräsentieren Knoten die einzelnen *Aktionen* einer Aktivität, sowie *Kontrollkonstrukte* und *Objektspeicher*, während Kanten die Abhängigkeiten von Vorgängerknoten ausdrücken und zwar in Form der Weitergabe von *Kontrolle* bzw. von

Daten. Der Inhalt einer Aktivität kann auch ohne den Rahmen der umgebenden Aktivität dargestellt werden

Knoten

Prinzipiell werden drei Knotenarten unterschieden – *Aktionsknoten*, *Kontrollknoten* und *Objektknoten*.

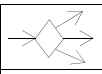
Aktionsknoten repräsentieren vordefinierte Aktionen

Aktionsknoten repräsentieren *vordefinierte UML-Aktionen*, die Eingaben empfangen und diese zu Ausgaben für andere Knoten verarbeiten können, oder andere Aktivitäten aufrufen. Aktionsknoten werden wie Aktivitäten in Form eines abgerundeten Rechtecks dargestellt, wobei der Name der Aktion als einziger Bestandteil, zentral innerhalb des abgerundeten Rechtecks plaziert wird. Aktionen werden im Detail in Kapitel 4.2.13 erläutert.

Kontrollknoten steuern Aktivitätsabläufe

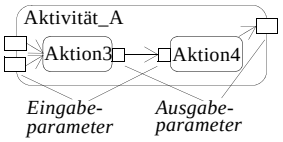
Kontrollknoten ermöglichen die Festlegung von Start und Ende einer gesamten Aktivität oder eines bestimmten Ablaufs einer Aktivität, die Spezifikation von alternativen Abläufen durch Verzweigungen bzw. Vereinigungen und die Modellierung von nebenläufigen Abläufen durch Parallelisierung bzw. Synchronisation. Tabelle 4–2 gibt einen Überblick über die verschiedenen Arten von Kontrollknoten und deren Notation. Für eine detaillierte Erläuterung sei auf die Kapitel 4.2.5 bis Kapitel 4.2.7 verwiesen.

Tab. 4–2
Arten von Kontrollknoten

Start und Ende von Abläufen		**Alternative Abläufe**		**Nebenläufige Abläufe**	
Initialknoten	●	Entscheidungs-knoten		Parallelisierungs-knoten	
Aktivitätsend-knoten		Vereinigungs-knoten		Synchronisierungs-knoten	
Ablaufend-knoten					

Objektknoten dienen als formale Ein- und Ausgabeparameter sowie als zentrale Puffer

Objektknoten stellen das Bindeglied zwischen der Verhaltensmodellierung in Form von Aktivitätsdiagrammen und der Strukturmodellierung, insbesondere in Gestalt des Klassendiagramms, dar. Objektknoten können Daten und Objekte aufnehmen. Sie dienen nicht nur als *formale Ein-* und *Ausgabeparameter* für eine bestimmte *Aktivität* oder *Aktion*, sondern können auch eine *zentrale Pufferfunktion* für Aktionen übernehmen. Trotz der Bezeichnung *Objekt*knoten können darin nicht nur Instanzen von *Classifiern* gespeichert werden, sondern ganz allgemein jegliche Art von Daten.



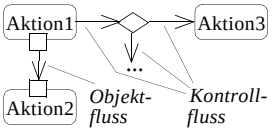
Ein Objektknoten wird prinzipiell als Rechteck dargestellt und kann im Falle einer Verwendung als Parameter direkt am Rahmen der Aktivität (überlappend) bzw. der Aktion (nicht überlappend in Form sogenannter *Pins*) »angeheftet« werden. Für Details zu Objektknoten siehe Kapitel 4.2.8.

Kanten

Kanten verbinden Knoten und legen mögliche Abläufe einer Aktivität fest

Allen Arten von Knoten einer Aktivität ist gemeinsam, dass sie durch *Kanten* miteinander verknüpft werden können, die zusammen mit den Kontrollknoten die zeitlich logische Reihenfolge der Aktionen und damit die möglichen Abläufe der gesamten Aktivität festlegen. Kanten ersetzen dabei das in UML1 zur Verknüpfung von Knoten verwendete Konzept der *Transitionen*, das ja eigentlich aus Zustandsdiagrammen stammt.

Kontroll- vs.



Eine Kante wird mit einer offenen Pfeilspitze dargestellt und kann einen Namen aufweisen. Es gibt zwei verschiedene Arten von Kanten, die sich nur aufgrund ihrer Verwendung, nicht jedoch in ihrer Notation unterscheiden. Je nachdem ob die Kante mit einem Objektknoten in Beziehung steht oder nicht, handelt es sich um eine *Objektflusskante* oder eine *Kontrollflusskante*.

Kontrollflusskanten drücken eine reine Kontrollabhängigkeit zwischen Vorgänger- und Nachfolgerknoten aus

Eine *Kontrollflusskante* erlaubt die Modellierung des Kontrollflusses und drückt so eine Abhängigkeit zwischen Vorgänger- und Nachfolgerknoten aus, d.h. der Nachfolgerknoten kann erst dann verarbeitet werden, wenn dessen direkte Vorgängerknoten die Kontrolle weitergeben. Kontrollflusskanten können Aktionsknoten direkt oder über entsprechende Kontrollknoten miteinander verbinden, Verknüpfungen zu Objektknoten sind nicht erlaubt.

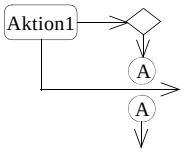
Objektflusskanten transportieren zusätzlich Daten und drücken dadurch auch eine Datenabhängigkeit zwischen Vorgänger- und Nachfolgerknoten aus

Eine *Objektflusskante* repräsentiert nicht nur, wie der Name suggeriert, den Objektfluss sondern ermöglicht auch den Transport reiner Daten. Zusätzlich zu dieser Transportfunktion für Daten bzw. Objekte wird auch eine Abhängigkeit zwischen Vorgänger- und Nachfolgerknoten ausgedrückt, da der Nachfolgerknoten erst dann verarbeitet werden kann, wenn der Vorgängerknoten die geforderten Daten bzw. Objekte weitergegeben hat. Eine Objektflusskante kann Objektknoten direkt oder über entsprechende Kontrollknoten miteinander verbinden. Eine direkte Verknüpfung von zwei Aktionsknoten über eine Objektflusskante ist jedoch nicht erlaubt.

Überwachungsbedingung (guard) bestimmt, ob Kontroll- und Datenfluss weiterläuft oder nicht

Sowohl für Kontroll- als auch für Objektflusskanten können *Überwachungsbedinungungen* in Form boolscher Ausdrücke spezifiziert werden. Eine Überwachungsbedingung legt fest, in welchem Fall die Kontrolle bzw. entsprechende Daten und Objekte über eine bestimmte Kante an den entsprechenden Nachfolgerknoten weitergereicht werden können.

Mechanismen zu Steuerung des Objektflusses
Fortsetzungsmarken (connectors)



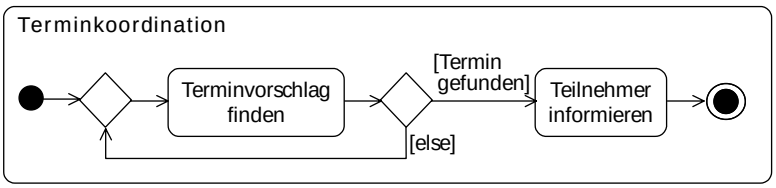
Schließlich existieren speziell für Objektflusskanten und Objektknoten eine Reihe von Mechanismen, die zusätzlich zum Kontrollfluss eine explizite Steuerung des Objektflusses ermöglichen. Diese umfassen beispielsweise die Festlegung der maximalen Kapazität einer Objektflusskante oder eines Objektknotens, sowie die Selektion und Transformation von Daten. Für Details dazu, siehe Kapitel 4.2.9.

Um bei komplexen Aktivitätsdiagrammen die Übersichtlichkeit zu gewährleisten, können Kreuzungen von Kontroll- und Datenflusskanten no-

tationstechnisch dadurch vermieden werden, dass sie unterbrochen und an anderer Stelle fortgeführt werden. UML bietet dazu die Möglichkeit, ähnlich wie bei Sequenzdiagrammen (siehe »*Fortsetzungsmarke*«, S. 297), sogenannte *Fortsetzungsmarken* zu definieren. Fortsetzungsmarken müssen immer paarweise auftreten, einmal mit einer eingehenden Kante und ein weiteres Mal mit einer ausgehenden Kante, sowie einen eindeutigen Namen aufweisen. Eine Fortsetzungsmarke wird als kleiner Kreis angeschrieben, in dem sich der Name der Fortsetzungsmarke befindet.

Abbildung 4–10 zeigt zwei Aktionen einer Aktivität *Terminkoordination* die versuchen, einen passenden Terminvorschlag für eine Reihe von Teilnehmern zu finden.

Abb. 4–10
Beispiel einer Aktivität zur Terminkoordination



Solange ein bestimmter Termin nicht bei allen Teilnehmern auf Zustimmung stößt, werden neue Termine vorgeschlagen Erst wenn ein für alle Teilnehmer passender Termin gefunden ist, werden die Teilnehmer entsprechend informiert und die Aktivität wird abgeschlossen.

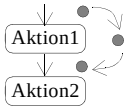
4.2.4 Token zur Koordination von Aktivitätsabläufen

Token als »virtueller Koordinationsmechanismus« mit dem Aktivitätsabläufe beschrieben werden können

Ein zentrales Konzept bei Aktivitätsdiagrammen stellen sogenannte *Token* dar. Es handelt sich dabei jedoch weder um ein eigenes Sprachkonzept, noch um ein eigenes Notationselement, sondern vielmehr um eine Art »virtueller Koordinationsmechanismus«, mit dem Abläufe einer Aktivität beschrieben werden können, um so z.B. eine exakte Vorgabe für die Implementierung der Aktivität zu liefern.

Token fliessen entlang der Kanten von Vorgänger- zu Nachfolgerknoten

Ein Token fließt entlang der Kanten von *Vorgängerknoten* zu *Nachfolgerknoten* und beschreibt somit einen möglichen *Ablauf* einer Aktivität zur Laufzeit. Durch die Spezifikation von Nebenläufigkeit sowie durch mehrfachen Aufruf einer Aktivität (siehe weiter unten) können sich auch mehrere Token zur gleichen Zeit in einem Aktivitätsdiagramm befinden, wobei jedes Token einem bestimmten Ablauf angehört.



Unterscheidung in Kontroll- und Datentoken

Token werden in *Kontroll-* und *Datentoken* unterschieden. Während ein Kontrolltoken eine reine Ausführungserlaubnis für den Nachfolgerknoten darstellt, transportiert ein Datentoken einen Datenwert oder eine Referenz auf ein Objekt (deswegen oftmals auch *Objekttoken* genannt). Falls diese Unterscheidung für das Verständnis nicht notwendig ist, wird in weiterer Folge nur von Token gesprochen oder aber in Kontroll- und Datento-

ken unterschieden. Die Ausführung eines Nachfolgerknotens kann auch durch die Verfügbarkeit entsprechender Datentoken alleine veranlaßt werden, weshalb Datentoken ebenfalls den Kontrollfluss steuern. Damit stellt ein Kontrolltoken eigentlich eine »degenerierte« Form eines Datentokens dar, der nur den Kontrollfluss steuert, jedoch keine Daten transportiert, während ein Datentoken beides leisten kann [Bock03].

Anbieten und Aufbewahren von Token sowie Auslösen der Verarbeitung durch Token

Im Allgemeinen müssen an allen eingehenden Kanten eines Knotens die erforderlichen Kontroll- bzw. Datentoken »angeboten« werden, d.h. zur Verfügung stehen, um dessen Ausführung zu ermöglichen. Sobald alle erforderlichen Token vorhanden sind, fließen diese in den Knoten ein und dieser kann ausgeführt werden. Die Token und damit auch die Kontrolle verbleiben solange bei diesem Knoten, solange dessen Ausführung dauert bzw. bis der nächste Knoten die Token akzeptiert. Eine Überwachungsbedingung kann beispielsweise die Weitergabe der Token verhindern, wodurch sich beim Vorgängerknoten mehrere Token ansammeln können. Token können jedoch nur in Aktions- und Objektknoten verweilen, Kontrollknoten geben die Token im Allgemeinen sofort weiter, Ausnahmen von dieser Regel werden bei den entsprechenden Konzepten behandelt.

Ausführung eines Aktionsknotens benötigt Zeit, Kontrollknotenausführung, sowie Tokenweitergabe ist zeitlos

Während die Ausführung eines Aktionsknotens eine endliche Zeitdauer in Anspruch nimmt, wird die Ausführung von Kontrollknoten, sowie die Weitergabe eines Tokens als zeitlos angesehen, da diese zur Laufzeit nicht tatsächlich erfolgen muss.

Optionen bei Mehrfachaufruf einer Aktivität – getrennte vs. gemeinsame Tokenverwal-

Eine Aktivität kann beliebig oft aufgerufen werden. Es wird dabei standardmäßig für jeden Aufruf eine neue Ausführung der Aktivität gestartet, sodass sich die einzelnen Abläufe bzw. Tokenflüsse gegenseitig *nicht beeinflussen*. Dies ist vergleichbar mit Betriebsystemprozessen, wobei jeder Prozess einen eigenen Adressraum verwendet. Als Alternative dazu kann durch Angabe des Schlüsselworts *«singleExecution»* im oberen Teil des Aktivitätsrechtecks festgelegt werden, dass im Falle eines Mehrfachaufrufs für alle Abläufe ein »gemeinsamer Adressraum« verwendet wird. Da dadurch gleichzeitig mehrere Abläufe, inklusive der zugehörigen Token gemeinsam verwaltet werden, können sich die verschiedenen Abläufe gegenseitig *beeinflussen*.

Unterschiede zu Petrinetzen

Abschließend sei betont, dass UML2 zwar auf dem Tokenkonzept von Petrinetzen aufbaut, sich aber im Detail von diesem unterscheidet. Für einen Vergleich zwischen Aktivitätsdiagrammen und Petrinetzen sei auf [Stör04] verwiesen.

4.2.5 Start und Ende von Aktivitäten und Abläufen

Start und Ende von Abläufen werden einerseits durch *Initialknoten* und andererseits durch *Aktivitätsend-* und *Ablaufendknoten* festgelegt.

Initialknoten

Initialknoten (initial node) kennzeichnet den Beginn eines Aktivitätsablaufs

Versorgt ausgehende Kanten mit Kontrolltoken

Aufbewahrung von Token erlaubt, da Überwachungs-bedingungen die Weiter-

Ein Initialknoten ist eine spezielle Art eines Kontrollknotens der dazu dient, den Beginn eines Ablaufs einer Aktivität zu kennzeichnen. Ein Initialknoten wird durch einen kleinen gefüllten Kreis dargestellt, ohne eingehende und mit beliebig vielen ausgehenden Kanten.

Sobald eine Aktivität aufgerufen wird, werden an allen ausgehenden Kanten eines Initialknotens Kontrolltoken zur Verfügung gestellt und somit der Ablauf der Aktivität gestartet.

Da *Überwachungsbedingungen* die Weitergabe der Kontrolltoken an die nachfolgenden Knoten beschränken können und gleichzeitig der Inititalknoten keinen Vorgängerknoten hat, in dem nicht akzeptierte Token gesammelt werden könnten, ist es im Gegensatz zu anderen Kontrollknoten bei Initialknoten erlaubt, Kontrolltoken solange aufzubewahren bis die nachfolgenden Knoten die Token akzeptieren.

Mehrere Initialknoten pro Aktivität erlaubt – ermöglicht

IWährend in UML1 der Start einer Aktivität eindeutig als *Pseudozustand* gekennzeichnet ist, kann eine Aktivität in UML2 auch *mehrere Initialknoten* besitzen, wodurch nebenläufige Abläufe gestartet werden können. Diese Möglichkeit ist z.B. für die Modellierung von Geschäftsprozessen von großem Nutzen. Im Falle eines Aufrufs der Aktivität werden die ausgehenden Kanten aller Initialknoten gleichzeitig mit entsprechenden Kontrolltoken versorgt.

Auch Aktivitäten ohne Inititalknoten erlaubt

Umgekehrt muss eine Aktivität nicht zwangsläufig einen Initialknoten besitzen. Auch über Parameter einer Aktivität (siehe Kapitel 4.2.8) können Token zur Verfügung stehen und somit Abläufe einer Aktivität gestartet werden. Darüber hinaus werden jene Knoten, die keine eingehenden Kanten (und damit auch keinen Initialknoten) aufweisen, bei Aufruf der Aktivität automatisch mit Kontrolltoken belegt. Im Unterschied zur Verwendung von expliziten Initialknoten ist dabei aber keine Angabe von Überwachungsbedingungen möglich.

Aktivitätsendknoten

Aktivitätsendknoten (activity final node) beendet alle Abläufe einer Aktivität

Mehrere Aktivitätsend-knoten pro Aktivität erlaubt

Keine Ausführung weiterer Aktionen

Ein Aktivitätsendknoten ist ein Kontrollknoten, der die Beendigung der Ausführung aller Abläufe einer Aktivität erzwingt. Ein Aktivitätsendknoten hat keine ausgehenden Kanten, kann aber mehrere eingehende Kanten aufweisen und wird durch einen kleinen Kreis, mit einem darin enthaltenen gefüllten Kreis dargestellt. Dies wird auch häufig als »bull's eye« bezeichnet.

Meist wird eine Aktivität nur *einen* Aktivitätsendknoten besitzen, sie kann aber auch *mehrere* aufweisen. In diesem Fall beendet jener Ablauf, der als erstes einen Aktivitätsendknoten erreicht, die gesamte Aktivität.

Das Beenden einer Aktivität bedeutet, dass zum Einen gerade aktive Aktionen gestoppt und aufgrund ihrer Atomarität allfällige Effekte rück-

gängig gemacht werden und dass zum Anderen keine weiteren Aktionen eventuell noch aktiver Abläufe der Aktivität ausgeführt werden. Wurde durch Aktionen anderes Verhalten eingebunden, so wird im Falle eines synchronen Aufrufs das eingebundene Verhalten ebenfalls beendet, während im asynchronen Fall das eingebundene Verhalten von der Beendigung der Aktivität nicht betroffen ist.

Kontrolltoken gelöscht, Datentoken an Ausgabeparametern der Aktivität nicht

Durch die Beendigung der Aktivität werden alle Kontrolltoken gelöscht. Datentoken, die bereits an den Ausgabeparametern der Aktivität anliegen, werden an den Aufrufer der Aktivität zurückgegeben, andere Datentoken werden gelöscht. Ausgabeparameter an denen zur Zeit der Beendigung der Aktivität noch kein Datentoken anliegt, werden mit einem sogenannten *»Null-Token«* – einer Art Nullwert – belegt.

Ende des Lebenszyklus' eines Objekts

Repräsentiert die Aktivität den Lebenszyklus eines Objekts, so wird durch Erreichen des Aktivitätsendknotens das Objekt selbst ebenfalls gelöscht.

Ablaufendknoten

Ablaufendknoten (flow final node) beendet einen Ablauf einer Aktivität

Oftmals ist es nicht wünschenswert, *alle* Abläufe einer Aktivität zu beenden, sondern nur einen *bestimmten* Ablauf. Dies ist nicht nur im Falle nebenläufiger Abläufe von Nutzen, sondern auch dann, wenn eine Aktivität als *«singleExecution»* spezifiziert ist, da in diesem Fall ein Aktivitätsendknoten alle Abläufe beenden würde und somit auch diejenigen, die durch mehrmalige Aufrufe der Aktivität initiiert wurden. Mit einem Ablaufendknoten kann spezifiziert werden, dass nur ein einziger dieser Abläufe beendet wird. Ein Ablaufendknoten wird durch einen kleinen Kreis mit einem darin enthaltenen Kreuz dargestellt und weist ausschließlich eingehende Kanten auf.

Abbildung 4–11 zeigt eine Aktivität zum Erstellen und Versenden von Einladungen zu einem Termin. Dazu werden die Einladungen einzeln gedruckt und eine nach der anderen adressiert, wobei parallel zum Adressieren in einem eigenen Ablauf die nächste Einladung gedruckt wird.

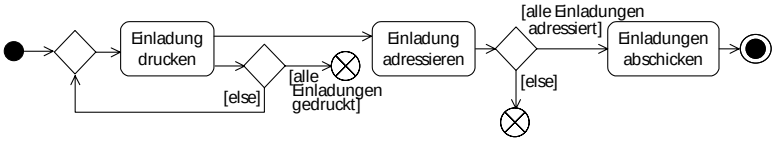


Abb. 4–11
Beispiel mit Initialknoten, Aktivitätsend- und Ablaufendknoten

Sind alle Einladungen gedruckt, wird der Ablauf zum Drucken der nächsten Einladung durch einen Ablaufendknoten beendet, die Adressierung kann jedoch weiterlaufen. Ist die Adressierung einer Einladung abgeschlossen, so wird der entsprechende Ablauf beendet, außer es handelt sich um die lezte Einladung. Sobald diese adressiert ist, können die Einladun-

gen abgeschickt werden und anschließend wird die gesamte Aktivität durch einen Aktivitätsendknoten beendet.

4.2.6 Alternative Abläufe

Zur Modellierung von alternativen Abläufen dienen *Entscheidungsknoten* und *Vereinigungsknoten*.

Entscheidungsknoten

Ein Entscheidungsknoten (decision node) definiert alternative Zweige und repräsentiert eine »Weiche« für den Tokenfluss

Ein Entscheidungsknoten ermöglicht es, einen Ablauf in beliebig viele alternative Zweige aufzuspalten, und kann damit als eine Art »Weiche« für die Token im Ablauf verstanden werden. Ein Token das den Entscheidungsknoten über die eingehende Kante erreicht, wird nur an genau eine ausgehende Kante weitergegeben.

Überwachungsbedingungen wählen den Zweig aus

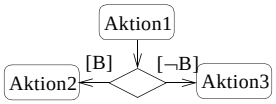
Die Entscheidung, an welcher ausgehenden Kante ein Token weiterfließt, wird mit Hilfe von *Überwachungsbedingungen* spezifiziert, die den einzelnen Kanten zugeordnet sind. In welcher Reihenfolge die Überwachungsbedingungen geprüft werden, ist nicht festgelegt.

Überwachungsbedingungen müssen wechselseitig ausschliessend sein, [else] ist vordefiniert Entscheidungsknoten zur Modellierung von

Es ist daher erforderlich, dass alle ausgehenden Kanten wechselseitig ausschließende Überwachungsbedingungen aufweisen. Eine ausgehende Kante kann mit der vordefinierten Bedingung *[else]* versehen werden, die dann zutrifft, wenn keine der anderen Bedingungen gültig ist.

Entscheidungsknoten können auch zur Modellierung von Schleifen verwendet werden, indem durch den Entscheidungsknoten das (neuerliche) Ausführen von Aktionen bestimmt wird. Eine weitere Möglichkeit zur Schleifenmodellierung stellen spezielle Schleifenknoten dar (siehe *Schleifenknoten*, S. 226).

Entscheidungsknoten



Ein Entscheidungsknoten wird durch eine Raute dargestellt und weist eine eingehende Kante und mehrere ausgehende Kanten auf. Alle Kanten eines Entscheidungsknotens müssen dabei entweder ausschliesslich Objektflüsse oder ausschliesslich Kontrollflüsse repräsentieren. Die Überwachungsbedingungen werden den jeweiligen Kanten in eckigen Klammern zugeordnet.

Entscheidungsverhalten (decision input behavior) ermöglicht detailliertere Spezifikation der Auswahlentscheidung an zentraler Stelle

Überwachungsbedingungen können die Auswahl des Zweiges auf Basis des eingegangenen Tokens treffen. Ist es notwendig, komplexere Auswahlentscheidungen zu spezifizieren und z.B. vor Auswahl des Zweiges noch eine Berechnung auf Basis des eingegangenen Tokens durchzuführen, kann ein sogenanntes *Entscheidungsverhalten* spezifiziert werden. Durch die Möglichkeit der zentralen Spezifikation beim Entscheidungsknoten selbst, können darüber hinaus auch redundante Evaluierungen einfacher Bedingungen vermieden werden, die andernfalls in Form von Über-

wachungsbedingungen an den ausgehenden Kanten modelliert werden müssten. Das Entscheidungsverhalten darf jedoch keine Nebeneffekte, d.h. Änderungen an Daten oder Objekten, aufweisen.

Ankunft von Token startet das Entscheidungsverhalten – Datentoken fungieren als Parameter

Entscheidungsverhalten wird immer dann ausgeführt, wenn der Entscheidungsknoten ein Token erhält, wobei ein eventuelles Datentoken den Eingabeparameter für das Entscheidungsverhalten darstellt, während im Falle eines Kontrolltokens das Entscheidungsverhalten ohne Parameter aufgerufen wird. Das Ergebnis des Entscheidungsverhaltens steht dann an den ausgehenden Kanten für die Evaluierung eventuell vorhandener Überwachungsbedingungen zur Verfügung.

Entscheidungsverhalten wird in einem Notizsymbol mit dem Schlüsselwort *«decisionInput»* beim entsprechenden Entscheidungsknoten annotiert.

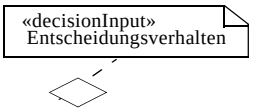
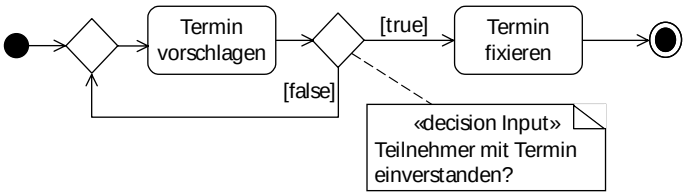


Abbildung 4–12 zeigt einen Entscheidungsknoten mit einem Entscheidungsverhalten, das prüft, ob die Teilnehmer mit einem bestimmten Terminvorschlag einverstanden sind.

Abb. 4–12
Beispiel mit Entscheidungs-knoten und



Das Ergebnis des Entscheidungsverhaltens wird von den Überwachungsbedingung zur Auswahl des Zweiges genutzt, wodurch entweder erneut ein weiterer Termin vorgeschlagen, oder aber der Termin fixiert wird.

Vereinigungsknoten

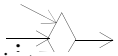
Ein Vereinigungsknoten (merge node) führt alternative (keine nebenläufigen!) Abläufe wieder zusammen

Token werden, sobald möglich, an den Nachfolgerknoten weitergereicht

Ein Vereinigungsknoten ermöglicht es, alternative Abläufe, die durch einen Entscheidungsknoten aufgespalten wurden, bei Bedarf wieder zu einem Zweig zusammenzuführen.

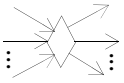
Sobald der Nachfolgerknoten zur Aufnahme von Token bereit ist, akzeptiert der Vereinigungsknoten den Token vom Vorgängerknoten und gibt diesen an den Nachfolgerknoten weiter.

Vereinigungsknoten



Für einen Vereinigungsknoten wird das gleiche Symbol wie für einen Entscheidungsknoten – eine Raute – verwendet. Ein Vereinigungsknoten besitzt mehrere eingehende Kanten und nur eine ausgehende Kante. Analog zu Entscheidungsknoten müssen dabei alle Kanten ausschließlich Objektflüsse oder ausschließlich Kontrollflüsse repräsentieren.

Kombinierter Entscheidungs-und

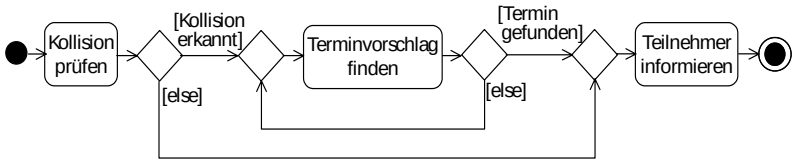


Schließlich besteht die Möglichkeit, durch ein und das selbe grafische Notationssymbol – die Raute – einen Vereinigungsknoten und einen Ent-

scheidungsknoten miteinander zu kombinieren, wodurch mehrere eingehende und mehrere ausgehende Kanten erlaubt sind.

Abbildung 4–13 zeigt eine Aktivität, die eine Erweiterung der Aktivität *Terminkoordination* aus Abbildung 4–10 vornimmt, indem in einer eigenen Aktion etwaige Kollisionen geprüft werden. Im Falle einer Kollision bzw. solange noch kein Termin gefunden wurde (erster Vereinigungsknoten), wird nach einem entsprechenden alternativen Terminvorschlag gesucht.

Abb. 4–13
Beispiel für Vereinigungsknoten



Falls keine Kollisionen erkannt wurden oder ein neuer Termin gefunden werden konnte (zweiter Vereinigungsknoten), werden die Teilnehmer über den Termin informiert.

4.2.7 Nebenläufige Abläufe

Zur Modellierung von nebenläufigen Abläufen dienen *Parallelisierungs-* und *Synchronisierungsknoten*.

Parallelisierungsknoten

Parallelisierungsknoten (fork node) zur Modellierung nebenläufiger Abläufe

Ein Parallelisierungsknoten ermöglicht es, einen Ablauf in beliebig viele nebenläufige Zweige aufzuspalten, die jedoch nicht notwendigerweise parallel ablaufen müssen, wie der Name suggeriert. Allein die Reihenfolge, in der die Zweige durchlaufen werden, ist beliebig.

Eingehende Token werden für alle ausgehenden Kanten dupliziert, sobald zumindest eine Überwachungsbedingung diese akzeptiert

Um eine Aufspaltung des Ablaufs in mehrere Zweige zu erreichen, werden die am Parallelisierungsknoten anliegenden Token für alle ausgehenden Kanten dupliziert. Dies erfolgt allerdings nicht automatisch sobald ein Token an der eingehenden Kante anliegt, sondern erst, wenn mindestens eine der ausgehenden Kanten unter Berücksichtigung eventuell vorhandener Überwachungsbedingung das Token akzeptiert. Ist dies der Fall, werden Token sofort auch für jene Kanten dupliziert, bei denen die Evaluierung der Überwachungsbedingungen erfolglos war.

Nichtakzeptierte Token werden aufbewahrt

Diese Token bleiben in der Reihenfolge ihrer Erzeugung (FIFO-Prinzip) erhalten, bis sie von den Überwachungsbedingungen akzeptiert werden. Dies stellt (neben Initialknoten) eine weitere Ausnahme von der Regel dar, dass Kontrollknoten keine Token aufbewahren dürfen.

Ein Parallelisierungsknoten wird als schwarzer Balken mit einer eingehenden Kante und mindestens zwei ausgehenden Kanten dargestellt. Analog zu Entscheidungsknoten müssen alle Kanten entweder ausschliesslich Objektflüsse oder ausschliesslich Kontrollflüsse repräsentieren.

Parallelisierungsknoten

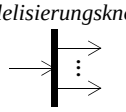


Abbildung 4–14 zeigt einen Parallelisierungsknoten, durch den im Anschluss an das Eintragen eines Termins gleichzeitig die entsprechenden Teilnehmer informiert werden und geprüft wird, ob ein weiterer Termin einzutragen ist.

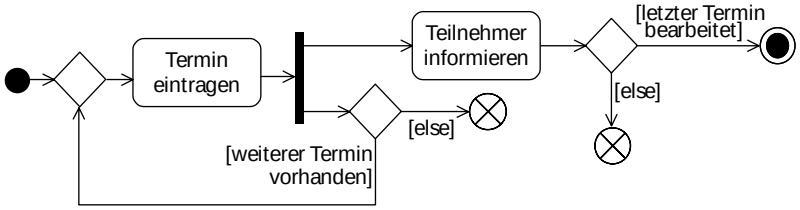


Abb. 4–14
Beispiel für einen Parallelisierungsknoten

Falls ja, wird der Termin eingetragen, falls kein weiterer Termin vorhanden ist, wird dieser Ablauf durch einen Ablaufendknoten terminiert, es können aber weiterhin Teilnehmer informiert werden. Sobald alle Teilnehmer eines Termins informiert sind, wird geprüft, ob dies der letzte zu bearbeitende Termin war. Ist dies der Fall, so kann die gesamte Aktivität beendet werden. Andernfalls wird nur dieser eine Ablauf beendet.

Synchronisierungsknoten

Synchronisierungsknoten (join node) führt nebenläufige Abläufe zusammen

Ein Synchronisierungsknoten agiert sozusagen als Gegenstück zum Parallelisierungsknoten, da eine Zusammenführung nebenläufiger Abläufe zu *einem* Ablauf erfolgt, indem die eingehenden Token entsprechend vereinigt werden.

Vereinigung der Token sobald an allen Kanten vorhanden

Die Vereinigung der Token erfolgt standardmäßig auf Basis einer vordefinierten, konjunktiven *Synchronisierungsbedingung*, d.h. sobald an allen eingehenden Kanten Token anliegen. Details der Vereinigung hängen davon ab, ob es sich um Kontroll- und/oder Datentoken handelt, bzw. an welchen Kanten die Token anliegen:

An einer Kante anliegende Kontrolltoken werden vereinigt

- In einem allerersten Schritt werden alle Kontrolltoken, die an *ein und der selben eingehenden Kante* anliegen, zu *einem* Kontrolltoken vereinigt.

Kontrolltoken verschiedener Kanten werden vereinigt und nur ein einzelnes Token weitergereicht

- Liegen Kontrolltoken auch an *verschiedenen Kanten* an, so werden diese ebenfalls zu *einem* Kontrolltoken vereinigt. Liegen keine Datentoken an, so wird dieses Kontrolltoken an der ausgehenden Kante angeboten.

Datentoken werden alle weitergereicht

- Liegen Datentoken an, so werden *alle* diese Datentoken auf der ausgehenden Kante angeboten. Dies erfolgt in der Reihenfolge, in der die Datentoken den Synchronisierungsknoten erreichen (FIFO-

Prinzip). Falls die Datentoken Objekte gleicher Identität beinhalten, werden diese standardmäßig zu einem Objekt vereinigt, bevor sie angeboten werden.

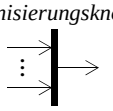
Bei Kontroll- und Datentoken werden nur die Datentoken weitergereicht

- Liegen Kontroll- und Datentoken an, werden nur die Datentoken weitergereicht.

Nichtakzeptierte Token gehen verloren

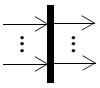
Token werden vom Nachfolgerknoten akzeptiert oder zurückgewiesen, wobei in diesem speziellen Fall ausnahmsweise zurückgewiesene Token verloren gehen.

Synchronisierungsknoten



Der Synchronisierungsknoten wird, analog zum Parallelisierungsknoten, als dicker schwarzer Balken mit mehreren eingehenden Kanten und genau einer ausgehenden Kante dargestellt. Anders als bei allen bisher behandelten Kontrollknoten ist es beim Synchronisierungsknoten möglich, Kontrollflüsse mit Objektflüssen zu synchronisieren. Dabei muss, sobald eine der eingehenden Kanten einen Objektfluss darstellt, auch die ausgehende Kante einen Objektfluss repräsentieren.

Kombinierter Parallelisierungs- und Synchronisierungsknoten



Analog zu Entscheidungs- und Vereinigungsknoten können auch Parallelisierungs- und Synchronisierungsknoten kombiniert und mit *einem* Notationselement – dem schwarzen Balken – dargestellt werden. In diesem Fall sind demnach mehrere ein- und ausgehende Kanten erlaubt.

Abbildung 4–15 zeigt eine Aktivität mit einem Parallelisierungsknoten, der im Anschluss an die erfolgreiche Prüfung einer Benutzerkennung, eine parallele Verarbeitung der Terminerfassung und der Teilnehmerzuordnung ermöglicht.

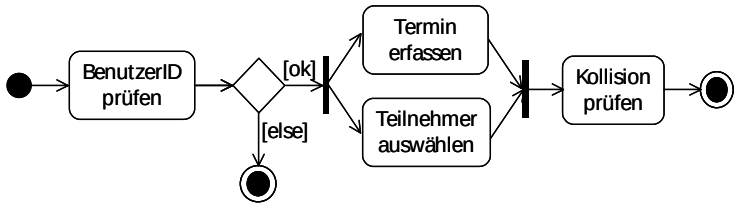
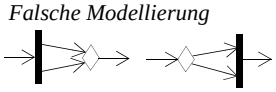


Abb. 4–15
Beispiel für einen Paralleli-sierungs- und einen

Erst wenn beide Aktionen vollständig durchgeführt sind, erlaubt der Synchronisierungsknoten die Ausführung der Nachfolgeraktion zur Kollisionsprüfung. Bei diesem Beispiel wurden bewusst zwei Aktivitätsendknoten verwendet, da die Aktivität sowohl bei einer falschen Benutzerkennung, als auch nach der Kollisionsprüfung beendet werden soll. Die beiden Aktivitätsendknoten könnten jedoch auch durch *einen* ersetzt werden, dem dann aber ein Vereinigungsknoten vorangestellt werden müsste. Ohne Vereinigungsknoten würde ein Modellierungsfehler vorliegen, da zwei Kanten in den Aktivitätsendknoten münden würden, wodurch dieser nur beim Anliegen von zwei Token aktiv werden könnte, was im vorliegenden Beispiel nicht eintreten kann.

Falsche Modellierung



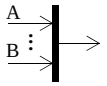
Eine weitere inkonsistente Situation kann sich ergeben, wenn nach einem Parallelisierungsknoten ein Vereinigungsknoten verwendet wird oder nach einem Entscheidungsknoten ein Synchronisierungsknoten. Parallele

Abläufe können nur durch einen Synchronisierungsknoten wieder zusammengeführt werden und alternative Abläufe nur durch einen Vereinigungsknoten.

Durch Angabe einer benutzerdefinierten *Synchronisierungsbedingung* kann auf bestimmte Kombinationen anliegender Token gewartet und erst dann mit der Abarbeitung fortgesetzt werden.

Synchronisierungsbedingung (join specification) zur Steuerung des Zeitpunkts der Tokenweitergabe

{joinSpec = A and B and X<Y}



Die Synchronisierungsbedingung ist eine boolsche Bedingung, die in geschwungenen Klammern angeschrieben wird und auf die eingehenden Kanten über deren Namen Bezug nehmen kann. Eine Synchronisierungsbedingung wird immer dann ausgewertet, wenn ein neues Token an einer der eingehenden Kanten anliegt. Neu hinzukommende Token unterbrechen eine gerade laufende Auswertung nicht und starten erst dann eine neue Auswertung, wenn die aktuelle beendet ist.

Abbildung 4–16 zeigt weitere Aspekte der Aktivität Terminkoordination. Parallel zum Finden eines Terminvorschlag wird nun eine Kontaktperson für den Termin ausgewählt.

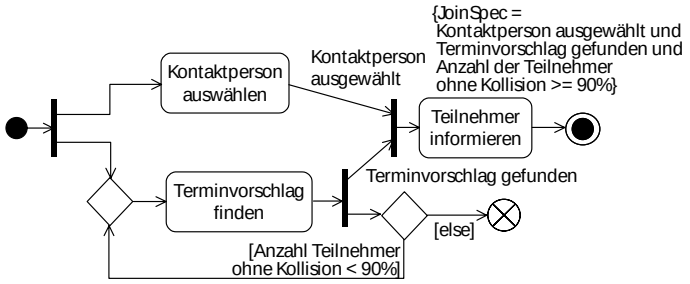


Abb. 4–16
Beispiel für eine Synchroni-

In einer entsprechenden Synchronisierungsbedingung wird festgelegt, dass die Synchronisierung beider Abläufe und damit die Ausführung der Nachfolgeraktion *Teilnehmer informieren* dann erfolgen kann, wenn die Kontaktperson ausgewählt ist und bei zumindest 90% der Teilnehmer keine Terminkollisionen auftreten. Im vorliegenden Beispiel könnte statt der Synchronisierungsbedingung auch ein Verzweigungknoten verwendet werden, der - ohne vorangehenden Parallelisierungsknoten - entweder wieder auf *Terminvorschlag finden* verzweigt, oder - anstatt in einen Ablaufendknoten zu münden - zum Synchronisierungsbalken vor der Aktion *Teilnehmer informieren* führt.

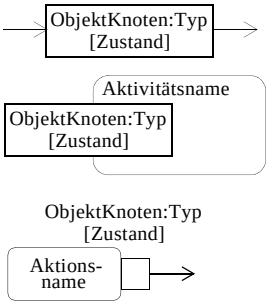
4.2.8 Arten von Objektknoten

Ein Objektknoten (object node) nimmt Datentoken auf und steht mit einem Objektfluss in Beziehung

Allen Objektknoten ist gemeinsam, dass Sie Datentoken – und damit Daten und Objekte – aufnehmen können und ausschliesslich durch Objektflüsse miteinander verbunden sind. Datentoken werden zwischen zwei Objektknoten oder zwischen einem Objektknoten und einer Aktion verschickt.

Inhalt ist Ergebnis einer Aktion und Eingabe für eine weitere Aktion

Angabe eines Typs für einen Objektknoten sowie einer Zustandseinschränkung ist optional



Unterscheidung verschiedener Arten von Objektknoten

Der Inhalt eines Objektknotens – die Datentoken – ist das Ergebnis der vorangegangenen Aktion bzw. kann von dieser verändert worden sein und ist die Eingabe für die nachfolgende Aktion.

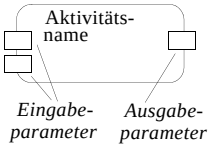
Für einen Objektknoten, dargestellt durch ein Rechteck, kann neben dem Namen optional ein *Typ*, durch Doppelpunkt vom Namen getrennt, angegeben werden. Dadurch kann ein Objektknoten nur Objekte des angegebenen Typs oder eines Subtyps aufnehmen. Da der Name in vielen Fällen bereits den Typ ausdrückt, wird meist nur der Name angegeben. Darüber hinaus kann ein Objektknoten auch auf einen bestimmten *Zustand* eingeschränkt werden, sodass er nur Token aufnehmen kann, die Objekte im entsprechenden Zustand referenzieren. Eine Zustandseinschränkung kann in eckigen Klammern unterhalb des Namens angegeben werden. Ist weder ein Typ noch eine Zustandseinschränkung spezifiziert, kann ein Objektknoten beliebige Daten und Objekte aufnehmen. Wie bereits erwähnt wird ein Objektknoten, im Falle einer Verwendung als Parameter, direkt am Rahmen der zugehörigen Aktivität (überlappend) bzw. der Aktion (nicht überlappend in Form sogenannter *Pins*) »angeheftet«.

Die im folgenden beschriebenen Arten von Objektknoten unterscheiden sich im wesentlichen darin, ob sie als Parameter fungieren oder als zentrale Puffer existieren und in welcher Form sie Token aufnehmen bzw. weitergeben.

Objektknoten als Parameter von Aktivitäten und Aktionen

Objektknoten, die einer Aktion oder Aktivität zugeordnet sind, fungieren als formale Ein- und Ausgabeparameter.

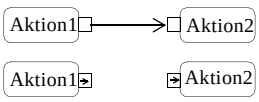
Ein- und Ausgabeparameter für Aktivitäten – Aktivitätsparameterknoten (activity parameter node)



Ein *Aktivitätsparameterknoten* ist ein Objektknoten der einer Aktivität zugeordnet ist und als formaler Ein- bzw. Ausgabeparameter fungiert. Seine Aufgabe besteht demnach darin, einer Aktivität Datentoken zur Verfügung zu stellen bzw. Datentoken an deren Aufrufer zurückzugeben. Daher muss der Typ des Aktivitätsparameterknotens auch jenem des Parameters des Aufrufers entsprechen. Je nachdem, ob ein Aktivitätsparameterknoten als Ein- oder Ausgabeparameter fungiert, darf er nur ausgehende Objektflüsse oder aber nur eingehende Objektflüsse aufweisen.

Ein- und Ausgabeparameter für Aktionen (Pins)

Implizite vs. explizite Kennzeichnung als Ein- und Ausgabepin



Formale Parameter von Aktionen werden als *Pins* bezeichnet. *Eingabepins*, ebenso wie *Ausgabepins*, werden als kleine Rechtecke dargestellt, die an Aktionsknoten angefügt werden. Bei Aktionen die anderes Verhalten aufrufen, kann bei den Pins auch die vollständige Spezifikation des entsprechenden Parameters des aufzurufenden Verhaltens verwendet werden.

Ist der Objektfluss zwischen Pins nicht durch entsprechende Kanten ersichtlich, dann kann die Kennzeichnung als Eingabe- bzw. Ausgabepin

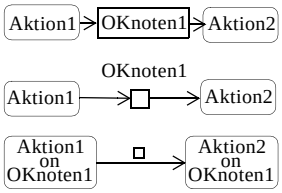
explizit durch kleine Pfeile im Pinsymbol erfolgen. Dabei zeigt bei Eingabepins die Pfeilspitze zum Aktionsknoten, während bei Ausgabepins die Pfeilspitze vom Aktionsknoten weg zeigt.

Obwohl die Pfeilrichtung der Objektflusskanten bzw. die in den Pins dargestellten Pfeile bereits anzeigen, ob es sich bei einem Pin um einen Eingabe- oder einen Ausgabepin handelt ist es üblich, wie auch bereits bei Ein- und Ausgabeparametern von Aktivitäten angemerkt, einen Eingabepin links bzw. oberhalb eines Aktionsknotens darzustellen, während ein Ausgabespin üblicherweise rechts bzw. unterhalb des Aktionsknotens dargestellt wird.

Notationskonvention - Eingababepins links bzw. oberhalb einer Aktion Ausgabepins rechts bzw. unterhalb einer Aktion

Falls der Typ des Ausgabepins einer Aktion jenem des Eingabepins der Nachfolgeraktion entspricht, können statt der Repräsentation in Form von zwei Pins weitere alternative Darstellungen verwendet werden. Erstens kann stattdessen ein einzelner Objektknoten in Form eines Rechtecks notiert werden. Zweitens kann statt des Objektknotenrechtecks ein kleines Pinrechteck verwendet werden und der Name des Objektknotens oberhalb angegeben werden. Und als dritte Variante ist es auch möglich, nur eine einzelne Objektflusskante vom Vorgänger- zum Nachfolgerknoten zu zeichnen, oberhalb ein kleines Pinrechteck anzugeben und auf den Namen des Pins zu verzichten, falls dieser bereits aus der Bezeichnung der beteiligten Aktionen hervorgeht.

Notationsvarianten für Pins



Für Ein- und Ausgabeparameter kann bei den entsprechenden Pins angegeben werden, welchen Effekt (*create*, *read*, *update* oder *delete*) eine Aktion auf die entsprechenden Daten und Objekte hat. Dabei können naturgemäß neu erstellte Daten und Objekte nur über einen Ausgabepin zur Verfügung gestellt werden, während Daten und Objekte, die von einer Aktion gelöscht werden sollen, an die Aktion durch einen Eingabepin übergeben werden.

Effekte einer Aktion – create, read, update und delete

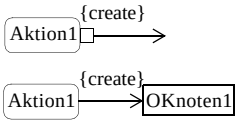


Abbildung 4–17 zeigt eine Aktion zur Terminerfassung, die einen Termin erzeugt und an einem Ausgabepin zur Verfügung stellt, sowie eine Aktion Teilnehmer auswählen, deren Ausgabepin einen Teilnehmer enthält.

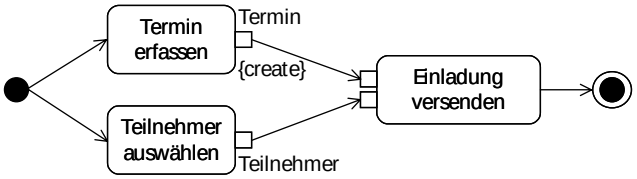


Abb. 4–17
Beispiel für Ein- und Ausgabepins

Die Nachfolgeraktion ist dafür zuständig, den über Eingabepins eingehenden Teilnehmern eine Einladung zum Termin zu senden.

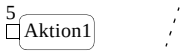
Konstanter Eingabewert – Wertepin

Wertepin (value pin) zur Übergabe konstanter Werte

Ein *Wertepin* ist eine spezielle Form eines Eingabepins. Dieser erlaubt es, für eine Aktion einen Wert, z.B. in Form einer Konstante, zur Verfügung zu stellen, ohne für dessen Erzeugung bzw. Weiterreichung eine eigene Aktion oder einen eigenen Objektfluss definieren zu müssen. Wertepins haben daher auch keine eingehenden Objektflüsse.

Wertepin startet nicht die Verarbeitung eines Knotens

Die Verarbeitung eines Knotens wird jedoch, im Gegensatz zu herkömmlichen Eingabepins, durch Wertepins alleine nicht gestartet. Der Wert wird erst dann an den Knoten weitergereicht, wenn dessen Verarbeitung zum Beispiel durch Kontrolltoken gestartet wird.



Wertepins werden wie Eingabepins dargestellt, zusätzlich wird der Wert neben dem Pinsymbol angegeben.

Pufferknoten

Zentrale Pufferung von Datentoken

Objektknoten können zentral für verschiedene Aktionen als *Pufferknoten* definiert werden. Derartige Objektknoten können zur Pufferung von Datentoken beliebig vieler Vorgängerknoten verwendet werden und diese beliebig vielen Nachfolgerknoten zur Verfügung stellen. Je nach Dauer der Speicherung der eingehenden Datentoken können *transiente Pufferknoten* (Schlüsselwort *«centralBuffer»*) und *persistente Pufferknoten* (Schlüsselwort *«datastore»*) unterschieden werden.

Transiente Pufferknoten (central buffer node) löscht weitergegebene Datentoken

Während ein *transienter Pufferknoten* gespeicherte Datentoken nicht weiter aufbewahrt, sobald sie an einen Knoten weitergegeben wurden, werden die Datentoken in einem *persistenten Pufferknoten* dupliziert, bevor sie weitergereicht werden.

Persistenter Pufferknoten (data store node) bewahrt sie auf und gibt Duplikate weiter

Ein persistenter Pufferknoten speichert somit alle eingehenden Datentoken, solange die Ausführung der Aktivität andauert, wodurch auf gespeicherte Datentoken bei Bedarf jederzeit zugegriffen werden kann.

Keine Mehrfachspeicherung identer Objekte
Explizites »Abholen« der Datentoken möglich

Ein eingehendes Datentoken, das ein Objekt referenziert, ersetzt bei einem persistenten Pufferknoten ein bereits vorliegendes Datentoken, das ein identes Objekt referenziert, d.h. idente Objekte liegen immer nur einmal vor. Dies ist ein wesentlicher Unterschied zu herkömmlichen Objektknoten, bei denen idente Objekte redundant gespeichert werden.

Selektionsverhalten ausgehender Objektflüsse (siehe Kapitel 4.2.9) kann dazu verwendet werden, um Datentoken aus dem persistenten Pufferknoten zu selektieren und diese nachfolgenden Aktionen über Eingabepins zur Verfügung zu stellen. Somit können Daten für Aktionen zu beliebigen Zeitpunkten »abgeholt« werden, anstatt darauf zu warten, dass die Datentoken automatisch weitergereicht werden, sobald sie vorliegen.

Abbildung 4–18 erweitert die Aktivität aus Abbildung 4–15 um zwei Pufferknoten.

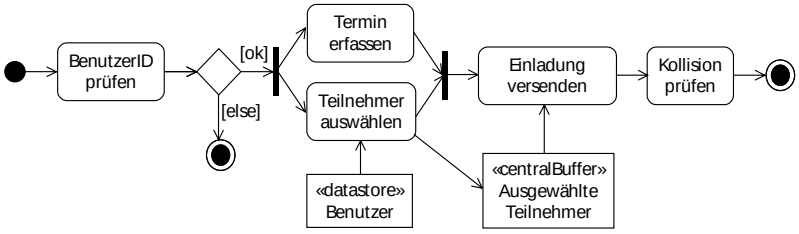


Abb. 4–18
Beispiel mit transienten und persistenten Pufferknoten

Der persistente Pufferknoten *Benutzer* speichert im Sinne einer Datenbank alle Benutzer des Terminkalenders, aus denen im Zuge der Teilnehmerauswahl bestimmte Benutzer selektiert werden. Ausgewählte Benutzer werden in einem transienten Puffer zwischengespeichert und erst für die Versendung von Einladungen wieder entnommen.

Ein- und Ausgabeparameter für Datenströme

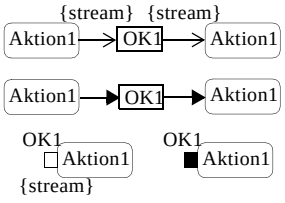
Ein- und Ausgabe von Datentoken während eine Aktion aktiv ist (streaming parameter)

Objektknoten die als Parameter von Aktionen verwendet werden, können zur Ein- und Ausgabe von Datenströmen verwendet werden. Derartige *Datenstromparameter* erlauben das Weiterreichen von Datentoken an eine Aktion bzw. aus einer Aktion heraus auch dann, wenn die Aktion gerade aktiv ist. Eine Aktion kann Datenströme sowohl nur als Eingabeparameter, als auch nur als Ausgabeparameter aufweisen oder als Ein- *und* Ausgabeparameter.

Start der Aktion, sobald Datentoken für zumindest einen Eingabeparameter vorhanden

Bevor die Aktion ausgeführt werden kann, müssen jedoch alle Eingabeparameter vorliegen, die nicht als Datenstromparameter gekennzeichnet sind. Weist das Verhalten ausschliesslich Eingabeparameter für Datenströme auf, so müssen zumindest Datentoken für einen dieser Parameter vorliegen.

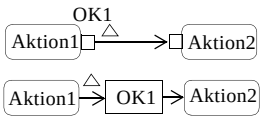
Datenstromparameter werden entweder mit *{stream}* gekennzeichnet, oder die ein- bzw. ausgehende Objektflusskante wird mit ausgefüllten Pfeilspitzen versehen, oder aber das entsprechende kleine Pinrechteck wird ausgefüllt dargestellt.



Ausgabeparameter für Ausnahmen (exception parameter)

Ausgabeparameter für Ausnahmen

Ausgabeparameter einer Aktion können als Ausnahmen gekennzeichnet werden. Die darauffolgende Aktion wird nur durchgeführt, falls das Auftreten einer Ausnahme durch das Vorhandensein eines solchen Ausnahmeparameters angezeigt wurde. Ausnahmeparameter werden nicht zusammen



mit gewöhnlichen Parametern am Ende einer Aktion weiter gegeben, sondern nur im Falle des Auftretens einer Ausnahme, wobei in diesem Fall dafür keine gewöhnlichen Parameter weitergegeben werden. Das gleiche gilt für Ausnahmeparameter von Aktivitäten, wobei hier im Fall einer Ausnahme alle Abläufe der Aktivität beendet werden. Die Kennzeichnung eines Ausnahmeparameters erfolgt durch ein kleines Dreieck an jener Objektflusskante, die von der Aktion, in der die Ausnahme ausgelöst werden kann, wegführt.

Gruppierung von Parametern – Parametersatz

Ein Parametersatz (parameter set) erlaubt die Gruppierung von Ein- bzw. Ausgabeparametern

Ein *Parametersatz* erlaubt die Zusammenfassung von Parametern, um somit alternative, einander ausschließende Gruppen von Ein- bzw. Ausgabewerten zu spezifizieren. Dies wird durch einen Rahmen um die zu gruppierenden Parameter ausgedrückt. Die durch einen Parametersatz zusammengefassten Parameter müssen alle entweder Eingabeparameter oder aber Ausgabeparameter sein. Ein Parameter kann dabei mehreren Parametersätzen zugeordnet sein, wobei jedoch unterschiedliche Parametersätze niemals genau dieselben Parameter aufweisen dürfen. Sobald zumindest ein Parametersatz existiert, müssen alle nicht durch einen Parametersatz zusammengefassten Eingabeparameter Datenstromparameter repräsentieren.

Nur ein ausgewählter Parametersatz ist jeweils für die Ausführung der Aktion relevant

Pro Ausführung der Aktion bzw. der Aktivität werden immer nur die Eingabeparameter *eines* der alternativen Parametersätze verwendet. Das bedeutet, dass nicht an allen *Pins* Datentoken anliegen müssen, um die Ausführung des Verhaltens zu starten, es reichen Datentoken an *allen* Pins *eines* Parametersatzes. Eine Aktion mit Parametersätzen für Ausgabeparameter kann wiederum nur Ausgabewerte für *einen* der Parametersätze generieren, d.h. nach Beendigung der Ausführung müssen nur an *allen* Pins *eines* Parametersatzes entsprechende Datentoken vorhanden sein.

Abbildung 4–19 zeigt eine Aktion zum Anlegen eines Terminkalendereintrags, der – abhängig von der Eintragsart – entweder einen Todo-Eintrag, einen Termin oder einen Serientermin erzeugt. Dafür sind jeweils unterschiedliche Informationen notwendig, die durch entsprechende Parametersätze – benannt nach der Eintragsart – gruppiert werden. Während für einen Todo-Eintrag Beginn- und Enddatum ausreichend sind, ist bei einem Termin zusätzlich der Ort und bei einem Serientermin Wiederholungsdauer und -frequenz erforderlich.

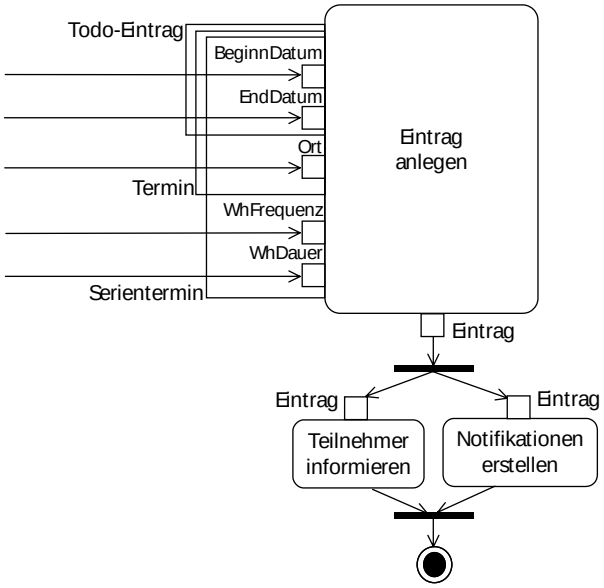


Abb. 4–19
Beispiel für Parametersätze

Ist der Kalendereintrag angelegt, werden die betroffenen Teilnehmern informiert und parallel dazu werden entsprechende Notifikationen erstellt, um die Teilnehmer kurz vor dem Beginndatum nochmals zu erinnern.

4.2.9 Steuerung des Objektflusses

Ein Objektfluss hat eine Transport- und eine Kontrollfunktion

Ein Objektfluss hat, wie bereits erwähnt, zwei Aufgaben. Zum Einen werden Datentoken von Vorgängerknoten zu Nachfolgerknoten transportiert, zum Anderen werden durch die Zurverfügungstellung der Datentoken Aktionen gestartet.

Ein Objektfluss verknüpft Aktionen nicht direkt, sondern über Objektknoten

Ein Objektfluss verknüpft *Objektknoten*, d.h. Aktionen können nicht direkt durch einen Objektfluss verbunden werden. Der Objektfluss kann dabei aber auch über Kontrollknoten geleitet werden.

Objektknoten bestimmen den Typ der zu transportierenden Objekte

Falls die beteiligten Objektknoten eines Objektflusses typisiert sind, bestimmen diese, welche Daten bzw. Objekte transportiert werden dürfen. Ist kein Typ angegeben, so kann ein Objektfluss Daten bzw. Objekte beliebiger Typen transportieren.

Weitergabe von Datentoken – Sortierung, Kapazi-tät, Gewicht, Filterung und

Im einfachsten Fall werden die Datentoken an der Objektflusskante unverändert zur Weiterverarbeitung angeboten. Dieses Standardverhalten für den Objektfluss kann jedoch durch verschiedene Sprachkonzepte beeinflusst werden, die beim Objektknoten bzw. bei der Objektflusskante selbst spezifiziert werden können. Dabei handelt es sich um die Spezifikation von *Reihenfolgen*, *Kapazitätsobergrenzen* und *Gewichten*, sowie *Selektions- und Transformationsverhalten.*

Reihenfolge der Tokenweitergabe

Die Reihenfolge, in der die Token weitergereicht werden, kann festgelegt werden

Die Reihenfolge, in der die Datentoken eines Objektknotens an eine ausgehende Objektflusskante weitergereicht werden, kann explizit festgelegt werden. Dabei können folgende Optionen innerhalb von geschwungenen Klammern beim Objektknoten spezifiziert werden:

FIFO (first in, first out)
{ordering = FIFO}

LIFO (last in, first out)
{ordering = LIFO}

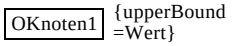
Geordnet
{ordering = ordered}

Ungeordnet
{ordering = unordered}

- Standardmässig ist die Weitergabereihenfolge als *FIFO* definiert, d.h. die Token werden in jener Reihenfolge weitergereicht, in der sie den Objektknoten erreichen.
- Sollen jene Token die *zuletzt* eingegangen sind als erste weitergereicht werden, so kann dies durch die Option *LIFO* angegeben werden.
- Wird eine benutzerdefinierte Reihenfolge gewünscht, so kann diese durch Angabe eines entsprechenden *Selektionsverhaltens* ausgedrückt werden (siehe weiter unten).
- Schliesslich kann auch angegeben werden, dass die Reihenfolge in der Token eingehen, keinen Einfluss auf die Reihenfolge hat, in der sie weitergereicht werden.

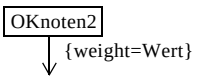
Kapazitätsobergrenze und Gewicht

Kapazitätsobergrenze (upper bound) limitiert die Anzahl der Token, die ein Objektknoten aufnehmen kann



Objektknoten können bezüglich der Anzahl von Token, die sie aufnehmen können, begrenzt werden. Die *Kapazitätsobergrenze* eines Objektknotens gibt in Form eines ganzzahligen Werts an, wieviele Token sich zu einem bestimmten Zeitpunkt maximal im Objektknoten befinden dürfen, standardmässig ist die Kapazität beliebig. Ist diese Obergrenze erreicht, so kann der Objektknoten keine weiteren Token mehr aufnehmen, diese verbleiben im Vorgängerknoten. Die Kapazitätsobergrenze der direkt über einen Objektfluss verbundenen Objektknoten muss gleich sein. Die Kapazitätsobergrenze wird beim Objektknoten in Form einer Einschränkung *(upperBound)* angegeben.

Gewicht einer Objektflusskante (weight)



Für eine Objektflusskante kann durch einen ganzzahligen Wert ein sogenanntes *Gewicht* angegeben werden. Das Gewicht gibt an, wie viele Token anliegen müssen, bevor diese an den Nachfolgerknoten weitergegeben werden. Es ist zu beachten, dass das Gewicht kleiner oder gleich der Kapazitätsobergrenze des Nachfolgerknotens sein muss. Das Gewicht wird bei der Objektflusskante ebenfalls in Form einer Einschränkung *(weight)* notiert.

Der in Abbildung 4–20 dargestellte Pufferknoten kann aufgrund der definierten Kapazitätsobergrenze maximal 20 Todo-Einträge aufnehmen.

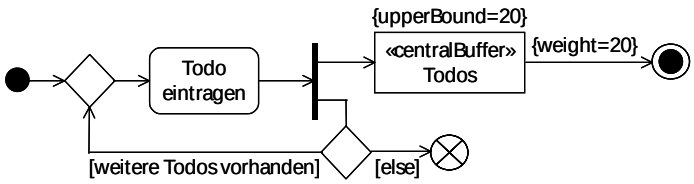


Abb. 4–20
Beispiel für Kapazitätsobergrenzen und Gewicht

Sollte diese Obergrenze erreicht werden, werden die Einträge – aus Überlastungsgründen des Benutzers – aufgrund des Gewichts der ausgehenden Kante an den Aktivitätsendknoten weitergeleitet und gelöscht.

Selektions- und Transformationsverhalten

Selektionsverhalten (selection behavior) wählt bestimmte Token zur Weitergabe aus

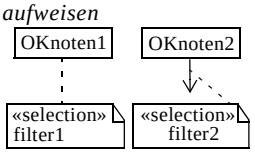
Durch die Definition eines *Selektionsverhaltens* können, im Unterschied zu einer Überwachungsbedingung, die nur über die Weitergabe aller Token entscheidet, bestimmte Token für die Weitergabe ausgewählt werden. Selektionsverhalten stellt damit eine Art von Tokenfilter dar und bestimmt damit implizit auch die Reihenfolge, in der die Token weitergereicht werden.

Objektknoten und Objektflusskanten können Selektionsverhalten aufweisen

Selektionsverhalten kann zum Einen beim *Objektknoten* selbst spezifiziert werden, wodurch an zentraler Stelle für alle Token bestimmt werden kann, welche dieser Token an *allen* ausgehenden Objektflusskanten angeboten werden. Zum Anderen kann Selektionsverhalten bei einer *Objektflusskante* definiert werden, wodurch spezifiziert wird, welche Token dem Objektknoten entnommen und über diese Kante an den Nachfolgerknoten weiterfliessen sollen.

Selektionsverhalten – z.B. in Form einer Aktivität – muss einen Eingabe- und einen Ausgabeparameter aufweisen

Selektionsverhalten kann durch eine beliebige Form der Verhaltensspezifikation definiert werden, zum Beispiel durch eine andere Aktivität, die jedoch nebeneffektfrei sein muss. Das Verhalten muss einen Eingabeparameter in Form einer Multimenge aufweisen und einen Ausgabeparameter, der das ausgewählte Token enthält. Die Token des Eingabeparameters müssen dabei vom Typ des Objektknotens oder aber eines Supertyps davon sein, der Ausgabeparameter vom Typ oder einem Subtyp.



Das Selektionsverhalten wird in einer Notiz mit dem Schlüsselwort *«selection»* dargestellt, die mit dem Objektknoten oder der Objektflusskante verbunden ist.

Transformationsverhalten (transformation behavior) transformiert Datentoken

Datentoken können vor ihrer Weitegabe nicht nur geordnet und gefiltert, sondern auch transformiert werden. So kann ein entsprechendes *Transformationsverhalten* den Datenwert eines Tokens austauschen, die Referenz auf ein Objekt durch eine andere austauschen oder aber ein eingehendes Datentoken in mehrere ausgehende Datentoken transformieren.

Beliebige Verhaltensspezifikation verwendbar

Transformationsverhalten kann analog zum Selektionsverhalten durch eine beliebige Form der Verhaltensspezifikation definiert werden und auch bezüglich Parameter und Nebeneffekten gelten diesselben Anforderungen wie beim Selektionsverhalten.

Transformationsverhalten wird in einer Notiz mit dem Schlüsselwort «*transformation*» an der Objektflusskante annotiert.

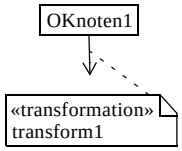
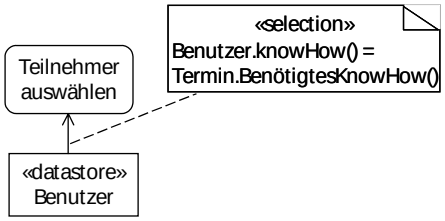


Abb. 4–21
Beispiel für Selektionsverhalten

Abbildung 4–21 zeigt ein einfaches Selektionsverhalten, mit dem beim Zugriff auf den persistenten Pufferknoten *Benutzer* relevante Benutzer für die endgültige Auswahl der Teilnehmer eines Termins vorselektiert werden.



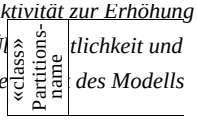
Selektionskriterium stellt dabei das Know-How des Benutzers dar, das mit dem für den Termin notwendigen Know-How übereinstimmen muss.

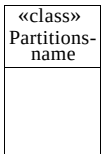
4.2.10 Partitionen

Eine Partition (activity partition) erlaubt die Gruppierung von Knoten und Kanten einer Aktivität nach bestimmten Kriterien

Eine *Partition* erlaubt die Gruppierung von Knoten und Kanten einer Aktivität auf Basis gemeinsamer Eigenschaften. Betrachtet man beispielsweise einen Geschäftsprozess, so könnten durch eine Partition alle Aktionen zusammengefasst werden, deren Ausführung z.B. im Verantwortlichkeitsbereich eines bestimmten *Standorts* liegt. Weitere Gruppierungskriterien zur Partitionierung einer Aktivität wären beispielsweise *Abteilungen*, *Geschäftsbereiche*, *Rollen*, *Ressourcen* oder *Subsysteme* (siehe auch [Jeck04c]). UML ist hinsichtlich der potentiell verwendbaren Gruppierungskriterien sehr offen – eine Partition kann beliebige Modellelemente umfassen, wobei jedoch bei deren Verwendung einige Regeln zu beachten sind, um die Konsistenz zu gewährleisten (siehe weiter unten).

»Schwimmbahnen«-Notation (swimlanes)
Aktivität zur Erhöhung der Übersichtlichkeit und Selektion des Modells





Partitionen beeinflussen nicht den Ablauf einer Aktivität, sondern stellen lediglich eine *logische Sicht* auf deren Bestandteile dar. Sie erhöhen die Übersichtlichkeit, erlauben eine rasche Erkennung der Verantwortlichkeitsbereiche und bringen damit auch mehr Detailinformation in das Modell ein.

Partitionen werden durch sogenannte »*Schwimmbahnen*« dargestellt, das sind zwei horizontale oder vertikale, parallel zueinandern verlaufende Linien, die am Kopfende ein Rechteck mit dem Partitionsnamen tragen. In einer solchen Schwimmbahn werden all jene Bestandteile der Aktivität

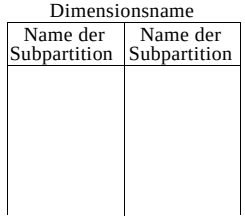
platziert, die der Partition zugeordnet sind. Der Partitionsname kann zusätzlich mit einem Schlüsselwort versehen werden, mit dem das Metamodellelement angegeben wird, auf dem die Partition basiert. Es sei an dieser Stelle darauf hingewiesen dass Kanten, die Knoten aus unterschiedlichen Partitionen miteinander verbinden, aus notationstechnischen Gründen nicht eindeutig einer Partition zugeordnet werden können.

Hierarchische, multidimensionale und externe Partitionen

Im Unterschied zu UML1 können Partitionen in UML2 auch *hierarchisch* angeordnet werden. Eine hierarchische Partition ist auf beliebig vielen Hierarchieebenen in weitere *Subpartitionen* unterteilt. Damit können Aktionen einer Aktivität beispielsweise nicht nur einem bestimmten Standort zugeordnet werden, sondern auf einer weiteren, darunterliegenden Hierarchieebene auch den entsprechenden zuständigen Abteilungen. Die Partition, die die Wurzel einer derartigen Verschachtelung repräsentiert, kann dabei als *Dimension* ausgezeichnet werden. Eine Dimension stellt einen »Diskriminator« dar, der eine disjunkte Partitionierung der Subpartitionen erfordert.

Hierarchische Partitionen zur Schachtelung auf verschie-denen

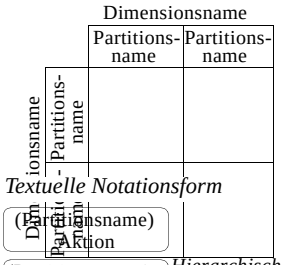
Betrachtet man die Notation für eine hierarchische Partition, so wird eine Schwimmbahn in weitere Bahnen unterteilt, sowie jede dieser Bahnen am Kopfende mit einem eigenen Rechteck inklusive Partitionsnamen und optional dem Schlüsselwort des verwendeten Metamodellelements versehen. Darüber wird der Name der Dimension und optional wiederum das Schlüsselwort des verwendeten Metamodellelements angegeben.



Multidimensionale Partitionen zur Zuordnung von Partitionen zu mehreren Dimensionen

Eine weitere Neuerung in UML2 stellt die Möglichkeit zur Definition *multidimensionaler Partitionen* dar. UML2 erlaubt die Verwendung mehrerer, orthogonaler Dimensionen auf der gleichen Hierarchieebene, wobei ein bestimmter Knoten innerhalb einer Dimension – wie auch bei hierarchischen Partitionen – ausschliesslich *einer* der Partitionen zugeordnet sein darf. So können z.B. die Partitionen *Produktion*, *Verkauf* und *Service* zu einer Dimension *Geschäftsbereich* zusammengefasst werden, während die Partitionen *Linz*, *Wien*, *Klagenfurt* zu einer, dazu orthogonalen Dimension *Standort* zusammengefasst werden können.

Die Einteilung in zwei orthogonale Dimensionen lässt sich ebenfalls in Form von Schwimmbahnen, die in Rasterform angeordnet werden, darstellen. Obwohl der UML-Standard mehr als zwei Dimensionen erlaubt, lässt sich dies mit Hilfe von Schwimmbahnen nicht mehr übersichtlich veranschaulichen.



Textuelle Notationsform

(Partitionsname) Aktion

Hierarchisch

(Partitionsname1:: Partitionsname2) Aktion

Multi-dimensional

(Partitionsname1, Partitionsname2) Aktion

Daher wird als Notationsalternative eine textuelle Angabe der Partition, innerhalb von runden Klammern, oberhalb des Namens des jeweiligen Aktionsknotens angeboten. Weist eine Aktion zwei oder mehr durch einen doppelten Doppelpunkt getrennte Partitionsnamen auf, so wird da-

durch eine hierarchische Partitionierung (Schachtelung von links nach rechts) ausgedrückt, handelt es sich um eine durch Kommata separierte Liste von Partitionsnamen, so wird dadurch eine multidimensionale Partitionierung dargestellt.

Knoten, die ausserhalb der Partitionsstruktur liegen, können in einer eigenen Partition zusammengefasst werden, die mit dem Schlüsselwort *«external»* gekennzeichnet wird.

.

Abb. 4–22
Beispiel für Partitionen

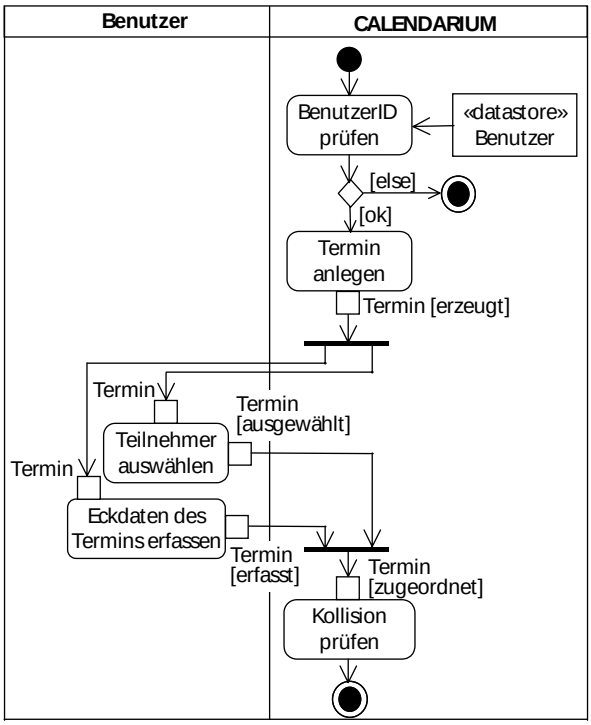


Abbildung 4–22 zeigt eine weitere Variante einer Terminkoordination, wobei in diesem Fall zwei Partitionen – Benutzer und Calendarium – zum Einsatz kommen und die beteiligten Aktionen entsprechend gruppiert wurden Zusammenhang zwischen Partitionen und Modellelementen

Partitionsdefinition muss der Definition des Modellelements entsprechen

Wie bereits erwähnt, kann eine Partition ein beliebiges UML Metamodellelement repräsentieren. Voraussetzung ist jedoch, dass eine Partition und die dadurch gruppierten Knoten und Kanten der Aktivität, der Definition des entsprechenden Modellelements nicht widersprechen dürfen. Im folgenden werden die zu beachtenden normativen Regeln für einige zentrale Metamodellelemente im Überblick dargestellt.

Classifier als Partition – repräsentiert Verhaltenskontext und muss bei hierarchischer Partitionierung Kompositionsbeziehungen aufweisen

Repräsentiert die Partition einen *Classifier*, so muss ein eventuell durch die Partition gruppierter Verhaltensaufruf in der Verantwortlichkeit des verwendeten *Classifiers* liegen, d.h. der *Classifier* muss den Kontext für den Verhaltensaufruf darstellen. Demzufolge müssen Prozeduraufrufe

und Ereignissignale zur Laufzeit eine Instanz des *Classifiers* als Ziel haben. Ist die Partition in einer anderen Partition geschachtelt, die ebenfalls einen *Classifier* repräsentiert, so muss zwischen den beiden *Classifiern* eine Kompositionsbeziehung bestehen oder der *Classifier* der eingebetteten Partition muss in dem übergeordneten *Classifier* definiert sein.

Rolle als Partition - gesamtes gruppiertes Verhalten auf Instanzen der Rolle beschränkt, Partitionsstruktur muss Classifier-Struktur entsprechen

Im Falle der Verwendung einer *Rolle* als Partition, liegt das gruppierte Verhalten in der Verantwortlichkeit der Instanzen, die die betreffende Rolle spielen. Es gelten dabei zunächst alle Regeln, die bereits beim *Classifier* beschrieben wurden, wobei Prozeduraufrufe und Ereignissignale als Ziel solche Instanzen verwenden müssen, die zur Laufzeit die entsprechende Rolle spielen. Ist die Partition hierarchisch strukturiert, so muss die Struktur der internen Struktur des betreffenden *Classifiers* entsprechen.

Instanz als Partition - gesamtes gruppiertes Verhalten auf diese eine Instanz beschränkt

Wird die Partition dazu verwendet, um eine *Instanz* zu repräsentiert, so ist das gruppierte Verhalten auf diese eine Instanz eingeschränkt, d.h. dass Prozeduraufrufe sowie Ereignissignale zur Laufzeit als Ziel die betreffende Instanz haben müssen.

Attribute als Partition - Werte als Subpartitionen

Eine Partition kann durch ein Attribut, und dessen Subpartitionen durch Werteausprägungen des Attributs, repräsentiert werden, wobei davon ausgegangen wird, dass die Werte zur Laufzeit für die entsprechenden Aktionen die sich in dieser Subpartition befinden, vorliegen.

4.2.11 Strukturierte Aktivitätsknoten

Ein strukturierter Aktivitätsknoten (structured activity node) dient zur Festlegung einer in sich abgeschlossenen

Ein *strukturierter Aktivitätsknoten* kann – analog zu Blöcken in Programmiersprachen – dazu verwendet werden, beliebige Teile einer Aktivität, d.h. Knoten und Kanten, zu einer in sich abgeschlossenen und strukturierten Ausführungseinheit zusammenzufassen. Strukturierte Aktivitätsknoten können wiederum weitere strukturierte Aktivitätsknoten enthalten, wodurch eine beliebige Schachtelung möglich ist.

Ein strukturierter Aktivitätsknoten wird wie ein herkömmlicher Knoten einer Aktivität behandelt

Ein strukturierter Aktivitätsknoten ist – wie der Name ausdrückt – selbst wiederum ein Knoten der Aktivität und wird in Bezug auf die Ablaufsemantik wie ein solcher behandelt. D.h. dass die in einem strukturierten Aktivitätsknoten befindlichen Aktionen erst dann gestartet werden, wenn alle erforderlichen Token am strukturierten Aktivitätsknoten anliegen. Ein strukturierter Aktivitätsknoten gibt die Kontrolle und damit die entsprechenden Token an nachfolgende (ausserhalb liegende) Knoten so lange nicht weiter, solange die Abarbeitung von Aktionen die innerhalb liegen andauert. Token befinden sich nur während der Ausführung innerhalb des strukturierten Aktivitätsknotens. Aktivitätsendknoten und Ablaufendknoten können auch in strukturierten Aktivitätsknoten modelliert werden.

Nur lokale Auswirkungen von Aktivitätsendknoten

Es ist dabei zu beachten, dass Aktivitätsendknoten innerhalb eines strukturierten Aktivitätsknotens nur lokale Auswirkungen haben, d.h. nur

den strukturierten Aktivitätsknoten beenden, nicht jedoch die gesamte Aktivität.

Ein- und Ausgabeparameter möglich

Ebenso unterstützen strukturierte Aktivitätsknoten *Ein- und Ausgabeparameter* in Form von *Pins*. Sind für den strukturierten Aktivitätsknoten Eingabepins vorhanden, so können die eingehenden Datentoken über entsprechende Objektflusskanten an die Knoten innerhalb der Aktivität weitergereicht werden.

Verknüpfung mit Knoten ausserhalb des strukturierten

Kanten ausserhalb des strukturierten Aktivitätsknotens können Vorgänger- oder Nachfolgerknoten innerhalb des strukturierten Aktivitätsknoten besitzen, allerdings nicht beides zugleich.

Merkmale im Unterschied zu nicht-strukturierten Knoten

Im Unterschied zu herkömmlichen, nicht-strukturierten Knoten haben strukturierte Aktivitätsknoten folgende Merkmale:

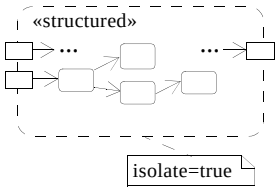
Namensraum für Bestandteile

- Ein strukturierter Aktivitätsknoten bildet für seine Bestandteile einen eigenen *Namensraum*. Damit können beispielsweise Variablen für den strukturierten Aktivitätsknoten definiert werden, die nur innerhalb des Knotens sichtbar sind.

Isolation konkurrierender Zugriffe

- Ein strukturierter Aktivitätsknoten kann zur *Isolation* von Aktionen verwendet werden. Dazu wird, falls zwei Aktionen gleichzeitig auf ein und dasselbe Objekt innerhalb und ausserhalb des strukturierten Aktivitätsknotens zugreifen, die Aktion ausserhalb verzögert, bis die gesamte strukturierte Aktivität in der die konkurrierende Aktion ausgeführt wurde, beendet ist. Um dies sicherzustellen, muss in einer Notiz am strukturierten Aktivitätsknoten das Metaattribut *isolate* auf *true* gesetzt werden.

Strukturierter, isolierter Aktivitätsknoten



Ein strukturierter Aktivitätsknoten wird als strichliertes, abgerundetes Rechteck mit dem Schlüsselwort «*structured*» dargestellt. Darin werden die Knoten und Kanten gezeigt, die Teile des strukturierten Aktivitätsknotens darstellen.

3 Arten strukturierter Aktivitätsknoten - Expansionsbereiche, Entscheidungs- und Schleifenknoten

UML stellt drei spezielle Formen strukturierter Aktivitätsknoten zur Verfügung, nämlich *Expansionsbereiche* zur Verarbeitung von Kollektionen, sowie Entscheidungs- und Schleifenknoten zur Realisierung von Kontrollkonstrukten, wodurch eine kompaktere Modellierung als mit Entscheidungsknoten möglich ist.

Expansionsbereich

Ein Expansionsbereich (expansion region) dient zur Verarbeitung von Kollektionen

Eine spezielle Form eines strukturierten Aktivitätsknotens ist ein sogenannter *Expansionsbereich*, der es erlaubt, die Elemente einer eingehenden Kollektion, *einzeln* zu verarbeiten, bzw. die Einzelelmente bei deren Ausgabe wieder zu einer Kollektion zusammenzufügen. Dabei können unterschiedliche Arten von Kollektionen verwendet werden, wie z.B. eine einfache Wertemenge, ein Multiset oder eine Liste von Werten.

Ein Expansionsknoten (expansion node) ist ein spezieller Objektknoten zum Aufgliedern und Zusammenfügen von Kollektionen Gleiche Kollektionsart aber unterschiedliche Elementtypen

Zur Aufgliederung in die einzelnen Elemente bei der Eingabe bzw. zum Zusammenfügen bei der Ausgabe werden spezielle Objektknoten – sogenannte *Expansionsknoten* – verwendet. Ein Expansionsbereich kann beliebig viele Expansionsknoten für die Ein- und Ausgabe von Kollektionen besitzen.

Besitzt ein Expansionsbereich mehrere eingehende Expansionsknoten, so müssen alle die *gleiche Kollektionsart* aufweisen. Der *Typ* der in den verschiedenen Kollektionen enthaltenen Elemente kann allerdings *unterschiedlich* sein.

Ein eigener Ablauf für alle Elemente einer bestimmten Position

Die Anzahl der Elemente einer Kollektion bestimmt die Anzahl der Abläufe innerhalb des strukturierten Aktivitätsknotens. Dabei werden die Elemente ein und derselben Position in verschiedenen Kollektionen in einem einzelnen Ablauf abgearbeitet.

Gleiche Anzahl an eingehenden Elementen pro Kollektion - jedoch beliebig viele ausgehende Elemente

Daher müssen alle eingehenden Kollektionen die *gleiche Anzahl von Elementen* aufweisen. Das Ergebniselement eines Ablaufs muss im Ausgabeexpansionsknoten an *dieselbe Position* platziert werden die auch das, die Ausführung startende Element im Eingabexpansionsknoten aufweist. Die Anzahl der Elemente der ausgehenden Kollektion kann von der Anzahl der Elemente der eingehenden Kollektion abweichen, wenn z.B. ein Ablauf für ein Element der eingehenden Kollektion keine Ausgabe produziert, wodurch der Expansionsbereich als eine Art Filter agiert.

Eingehende Objektflüsse ohne Expansionsknoten

Existieren Objektflüsse, die ausserhalb des Expansionsbereichs liegende Objektknoten direkt, d.h. ohne »Umweg« über einen Expansionsknoten, mit Objektknoten innerhalb des Expansionsbereichs verbinden, so werden für diese bei jedem Ablauf die *gleichen Eingabewerte* zur Verfügung gestellt.

Abarbeitungsreihenfolge der einzelnen Abläufe

Für einen Expansionsbereich kann festgelegt werden, in welcher Reihenfolge die einzelnen Abläufe abzuarbeiten sind. Dabei werden drei verschiedene Möglichkeiten unterschieden und durch entsprechende Schlüsselwörter gekennzeichnet:

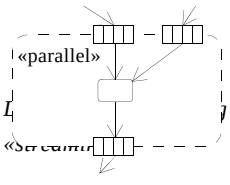
- Bei einer *parallelen Abarbeitung* können die einzelnen Abläufe eines Expansionsbereichs parallel bearbeitet werden.

Parallele Abarbeitung «parallel»

- Bei einer *iterativen Abarbeitung* werden die einzelnen Abläufe des Expansionsbereich nacheinander, gemäss der Position der Elemente im Eingabeexpansionsknoten, abgearbeitet. Ein Ablauf muss vollständig beendet sein, bevor der nächste Ablauf gestartet werden kann.

Iterative Abarbeitung Expansionsbereich

- Bei einer *Datenstromverarbeitung* werden die einzelnen Elemente des Eingabeexpansionsknotens einer Ausführung des Expansionsbereichs als Datenstrom zur *einmaligen Abarbeitung* zugeführt.



Explizite Kennzeichnung von Eingabe- und Ausgabeexpansionsknoten

Ein Expansionsbereich wird gemäss der Notation strukturierter Aktivitätsknoten als abgerundetes Rechteck mit strichlierter Linie dargestellt, das

Schlüsselwort zur Festlegung der Abarbeitungsreihenfolge der Abläufe wird links oben angegeben.

Ein Expansionsknoten wird als kleines Rechteck überlappend mit den Rändern des strukturierten Aktivitätsknoten dargestellt. Das Rechteck wird durch eine beliebige Anzahl vertikaler Linien unterteilt, um eine Kollektion von Elementen zu symbolisiert. Die Unterscheidung in Eingabe- und Ausgabeexpansionsknoten erfolgt implizit durch die eingehenden bzw. ausgehenden Objektflüsse ausserhalb des Expansionsbereichs, kann aber auch explizit, analog zu Pins, durch kleine Pfeile innerhalb der Abschnitte des Expansionsknotenrechtecks erfolgen (siehe *Objektknoten als Parameter von Aktivitäten und Aktionen*, S. 210).

Expansionsbereich mit nur einer Aktion

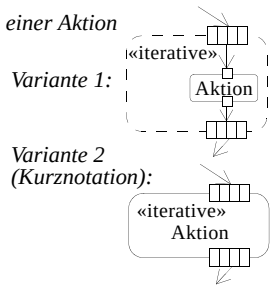
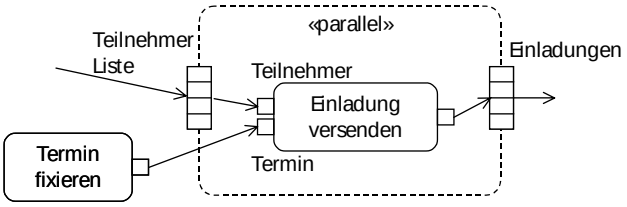


Abb. 4–23
Beispiel für einen Expansionsbereich

Besteht der Expansionsbereich aus nur einer Aktion, so kann als Kurzform anstatt eines strukturierten Aktivitätsknotens, ein Aktionsknoten benutzt werden, mit dem entsprechenden Abarbeitungsschlüsselwort und den Expansionsknoten an den Rändern.

Abbildung 4–23 zeigt einen parallelen Expansionsbereich zur Information der Teilnehmer eines Termins. Dazu wird zum Einen die Teilnehmerliste über einen Expansionsknoten an einen Eingabepin der Aktion weitergereicht und zum Anderen der fixierte Termin direkt an einen weiteren Eingabepin übergeben.



Als Ausgabe liefert der Expansionsbereich eine Liste von benachrichtigten Teilnehmern.

Konditionalknoten

Ein Konditionalknoten (conditional node) dient zur Auswahl aus einer Reihe alternativer Zweige

Ein *Konditionalknoten* ist ein spezieller strukturierter Aktivitätsknoten, der es im Sinne einer *if-then-else-Anweisung* erlaubt, aus alternativen Zweigen einen bestimmten Zweig auszuwählen. Ein Konditionalknoten stellt aufgrund seines blockorientierten Charakters das programmiersprachenspezifische Pendant zu einem *Entscheidungsknoten* dar, der aufgrund seiner ablauforientierten Natur eher dem Bereich der Prozessmodellierung zuzuordnen ist (siehe *Entscheidungsknoten*, S. 204). Gerade bei einer Vielzahl von Alternativen bringt ein Konditionalknoten durch seine Kompaktheit Vorteile.

Testbereiche (test sections) dienen zur Bedingungsprüfung, Ausführungsbereiche (body sections) zur Anweisungsausführung

Ein Konditionalknoten besteht aus ein oder mehreren *Testbereichen* (*if* bzw. *else if*), die jeweils eine Alternative repräsentieren und die zur Bedin-

gungsprüfung notwendigen Knoten und Kanten enthalten. Die *Ausführungsbereiche* (*then* und *else*) eines Konditionalknotens enthalten jene Knoten und Kanten, die bei erfüllter, bzw. im Falle eines *else* bei nichterfüllter Bedingung, durchlaufen werden.

Token starten die Bedingungsprüfung, die ein boolsches Ergebnis liefert

Damit im Testbereich mit der Bedingungsprüfung begonnen werden kann, müssen an allen eingehenden Kanten des Konditionalknotens Token anliegen. Das Ergebnis der Bedingungsprüfung ist ein boolscher Wert.

Mehrere if-Testbereiche

Sind mehrere *if*-Testbereiche spezifiziert, so ist die Reihenfolge der Bedingungsprüfung unbestimmt und eine einzige erfolgreiche Prüfung führt sofort zur Ausführung des entsprechenden Anweisungsteils und zum Abbruch aller übrigen Prüfungen. Ist keine der Bedingungen erfüllt, wird der Konditionalknoten beendet ohne Ausgaben zu produzieren, es sei denn dass ein *else*-Zweig existiert, der in diesen Fällen Abhilfe schafft.

else if-Testbereich

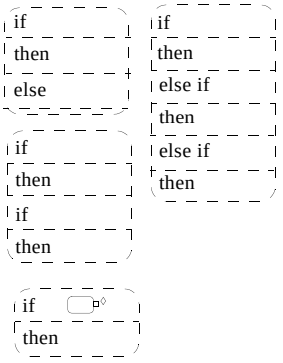
Bei einem *else if*-Testbereich wird die Bedingung erst dann geprüft, wenn die Evaluierung der Bedingung des *if*-Testbereichs nicht erfolgreich war. Ansonsten entspricht ein *else if*-Testbereich prinzipiell einem *if*-Testbereich, es werden dadurch jedoch mehrere Prüfungen ineinander geschachtelt und damit die Reihenfolge der Prüfungen festgelegt.

Nach Abarbeitung der Knoten im Ausführungsbereich ist die Aktivität beendet

Bei erfolgreicher Bedingungsprüfung wird der Ausführungsbereich gestartet, indem an allen darin enthaltenen Knoten die keine eingehenden Kanten besitzen, Kontrolltoken zur Verfügung gestellt werden. Sind alle darin enthaltenen Knoten abgearbeitet, ist auch die Ausführung des Konditionalknotens beendet. Werte, die im Testbereich oder im Ausführungsbereich generiert werden, können über Objektknoten als Ausgaben des Konditionalknotens an dessen Nachfolgerknoten weitergegeben werden.

»Inoffizielle« Notation für Konditionalknoten - Kombinationsbeispiele

Der aktuelle UML-Standard sieht keine Notation für Konditionalknoten vor, dies wird voraussichtlich erst mit der UML-Version 2.1 der Fall sein. Es gab allerdings in früheren Versionen des Standards eine sehr einfache Notation, die sich in zahlreichen UML-Büchern wiederfindet und auch in diesem Buch zur grafischen Illustration von Konditionalknoten verwendet werden soll. Ein Konditionalknoten wird dabei als Spezialform eines strukturierten Aktivitätsknotens durch ein strichliertes Rechteck mit abgerundeten Ecken dargestellt, wobei die einzelnen Bereiche durch strichlierte Linien voneinander getrennt und mit den Bezeichnungen der Testbereiche (*if* und *else if*) und der Ausführungsbereiche (*then* und *else*) versehen werden.



Ein Pin kann in einem Testbereich durch eine kleine Raute ausgezeichnet werden die angibt, dass dieser Pin in der Evaluierung der Bedingung verwendet wird. Wird die Raute bei einer Aktion ohne Pin verwendet, drückt dies aus, dass diese Aktion den endgültigen Wert der Bedingung berechnet (der jedoch nicht explizit angegeben wird).

Schleifenknoten

Ein Schleifenknoten (loop node) dient zur kompakten Modellierung von Schleifen

Schleifen können in Aktivitätsdiagrammen durch Entscheidungsknoten nur wenig intuitiv modelliert werden, sodass eine Schleife nicht immer auf den ersten Blick erkannt werden kann. Eine Alternative dazu stellt ein eigener *Schleifenknoten* dar, eine weitere spezielle Form eines strukturierten Aktivitätsknotens. Analog zu Entscheidungsknoten stellt auch der Schleifenknoten das programmiersprachenspezifische Pendant zu einem Entscheidungsknoten dar.

3 Bestandteile: Initialisierungsbereich (setup section), Testbereich (test section), sowie Ausführungsbereich (body section)

Ein Schleifenknoten besteht aus drei Teilbereichen - einem *Initialisierungsbereich*, einem *Testbereich* und einem *Ausführungsbereich*. Die Ausführung des Schleifenknotens bzw. der einzelnen Bereiche kann gestartet werden, wenn an allen eingehenen Kontroll- und Objektflusskanten Token anliegen. Für alle Knoten in dem jeweiligen Bereich, die keine eingehenden Kanten von ausserhalb des Bereichs aufweisen, werden automatisch Kontrolltoken zur Verfügung gestellt. Die Ausführung eines Bereichs muss jeweils beendet sein (d.h. alle Knoten müssen ausgeführt sein, die keine Nachfolger im jeweiligen Bereich haben), bevor der nächste Bereiche ausgeführt werden kann.

Funktionsweise der drei Bereiche

Der Initialisierungsbereich ermöglicht es, Aktionen auszuführen bevor die Schleife durchlaufen wird. Der Testbereich spezifiziert die Schleifenbedingung und muss somit schlussendlich einen boolschen Wert erzeugen der angibt, ob der Ausführungsbereich (erneut) durchlaufen wird. Der Ausführungsbereich spezifiziert schliesslich die Aktionen, die bei jedem Schleifendurchlauf ausgeführt werden sollen.

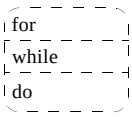
Abweisende und nicht-abweisende Schleifen

Mit einem Schleifenknoten kann sowohl eine *abweisende (while-) Schleife* realisiert werden, d.h. der Testbereich wird bereits vor dem Ausführungsbereich durchlaufen, als auch eine nicht-abweisende (do-while-) Schleife, d.h. der Testbereich wird erst nach der ersten Ausführung des Ausführungsbereichs durchlaufen.

Schleifenvariablen

Schleifen können Schleifenvariablen aufweisen, die zunächst mit Eingabewerten initialisiert werden, während der Ausführung der Schleife die aktuellen Werte der Schleifenvariablen beinhalten und am Ende der Schleife in Ausgabepins kopiert werden.

»Inoffizielle« Notation für Schleifenknoten



Bezüglich der Notation für Schleifenknoten gilt dasselbe wie für Konditionalknoten. Gemäss der »inoffiziellen« Notation wird ein Schleifenknoten als Spezialform eines strukturierten Aktivitätsknotens durch ein strichliertes Rechteck mit abgerundeten Ecken dargestellt, wobei die einzelnen Bereiche durch strichlierte Linien voneinander getrennt und mit entsprechenden Bezeichnungen für den Initialisierungsbereich -(*for*), den Testbereich (*while*) und den Schleifenrumpf (*do*) versehen werden.

In Abbildung 4–24 wird für die Aktivität *Terminkoordination* anstatt der in Abbildung 4–13 verwendeten ablauforientierten Notation ein Kon-

ditionalknoten mit einem darin geschachtelten Schleifenknoten verwendet.]

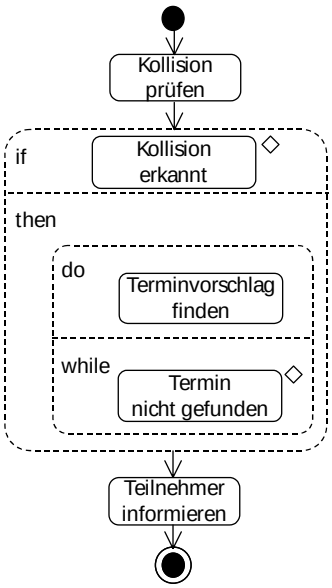


Abb. 4–24
Beispiel für einen Konditional- und einen Schleifenknoten

4.2.12 Ausnahmebehandlung und Unterbrechungsbereich

Vordefinierte Ausnahmen, beispielsweise durch das Laufzeitsystem

Während der Ausführung von Knoten können Ausnahmen auftreten, die die vorgesehene Abarbeitung beeinträchtigen. Solche Ausnahmen können einerseits durch das Laufzeitsystem vorgegebene sein, wie zum Beispiel Zugriffe auf einen Feldindex ausserhalb des definierten Bereichs oder eine Division durch Null. UML definiert allerdings nicht, welche vordefinierten Ausnahmen bei den verschiedenen Aktionen auftreten können.

Benutzerdefinierte Ausnahmen - RaiseExceptionAction

Andererseits können Ausnahmen auch explizit durch den Modellierer definiert werden, um Situationen die vom Regelfall abweichen aufzuzeigen. Diesem Zweck dient die vordefinierte Aktion *RaiseExceptionAction*. Diese empfängt als Eingabeparameter ein Objekt, dessen Typ die Ausnahme die ausgelöst werden soll, bestimmt. Ausnahmen sind Objekte eines beliebigen Modellelements, das auf einem *Classifier* basiert, wodurch Ausnahmen z.B. in Form einer Generalisierungshierarchie organisiert werden können.

Ablauf beim Auftreten einer Ausnahme

Tritt eine Ausnahme auf, so kann die »normale Ausführung« nicht fortgesetzt werden. Die Aktivität bzw. der strukturierte Aktivitätsknoten, der die Aktion welche die Ausnahme ausgelöst hat direkt umfasst, wird sofort beendet. Wird hingegen eine aufgetretene Ausnahme explizit abgefangen und behandelt, kann danach wieder zum »normale Ablauf« zurückgekehrt werden. Abhängig von der Art der Ausnahme, also dem Typ des

Ausnahmeobjekts, kann unterschiedliches Verhalten zur Behandlung der Ausnahme eingebunden werden.

Behandlung einer Ausnahme durch dezidierten Ausnahmebehandlungsknoten (exception handler)

Eine Ausnahmebehandlung wird durch einen eigenen *Ausnahmebehandlungsknoten* definiert, wobei ein solcher Knoten mehrere Typen von Ausnahmen behandeln kann. Ein ausführbarer Knoten (d.h. eine Aktion oder eine strukturierte Aktivität) für den eine Ausnahme definiert ist, wird als *geschützter Knoten* bezeichnet.

Finden passender Ausnahmebehandlungsknoten

Tritt eine Ausnahme auf, so wird nach einem passenden Ausnahmebehandlungsknoten gesucht. Dabei muss der Typ der Ausnahme gleich dem Ausnahmetyp sein der im Ausnahmebehandlungsknoten spezifiziert ist, oder aber ein Subtyp davon. Wird ein passender Ausnahmebehandlungsknoten gefunden, so wird dieser ausgeführt, wobei das Ausnahmeobjekt den Eingabeparameter darstellt.

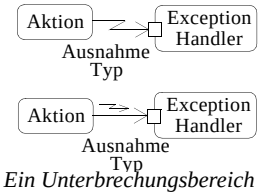
Der Ausnahmebehandlungsknoten substituiert den geschützten Knoten und hat daher keine eigenständigen ein- oder ausgehenden Kanten

Der Ausnahmebehandlungsknoten besitzt allerdings keine ausgehenden Kontroll- oder Objektflüsse. Die Ergebnisse des Ausnahmebehandlungsknotens werden nach dessen Ausführung, anstelle der Ergebnisse des geschützten Knotens, zur Verfügung gestellt - d.h. der Ausnahmebehandlungsknoten »substituiert« den geschützten Knoten. Die Ausführung kann danach fortgesetzt werden.

Propagierung von Ausnahmen

Existiert keine entsprechende Ausnahmebehandlung für den Ausnahmetyp, so wird die betroffene Aktion beendet und die Ausnahme nach aussen propagiert, d.h. es wird in der umgebenden Aktivität bzw. dem umgebenden strukturierten Aktivitätsknoten nach passenden Ausnahmebehandlungen gesucht. Wird trotz Propagierung bis auf die oberste Ebene keine passende Ausnahmebehandlung gefunden, so ist das Verhalten undefiniert.

Zwei Notationsvarianten



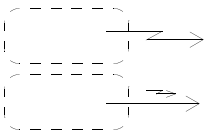
Ein Ausnahmebehandlungsknoten wird - da es sich um einen Aktionsknoten handelt - durch ein Rechteck mit abgerundeten Ecken dargestellt. Der geschützte Knoten verweist über einen blitzartigen Pfeil auf den Ausnahmebehandlungsknoten, wobei der Pfeil mit dem Typ der Ausnahme versehen wird. Als Alternative kann eine herkömmliche Kante verwendet werden, die mit einem kleinen blitzartigen Pfeil annotiert wird.

Ein Unterbrechungsbereich (interruptible activity region) ist eine Gruppe von Knoten und Kanten, deren Ausfüh-rung gemeinsam unter-brochen werden kann

Ausnahmebehandlungsknoten werden häufig gemeinsam mit sogenannten *Unterbrechungsbereichen* verwendet. Ein Unterbrechungsbereich gruppiert Knoten und Kanten einer Aktivität, deren Ausführung gemeinsam, z.B. durch eine Ausnahme, unterbrochen werden kann. Eine Unterbrechung wird dabei durch das Fliessen eines Tokens über eine *Unterbrechungskante* angezeigt, die von einem Knoten des Unterbrechungsbereichs zu einem ausserhalb liegenden Knoten derselben Aktivität führt. Einem Unterbrechungsbreich können eine oder mehrere *Unterbrechungskanten* zugeordnet sein. Verlässt ein Token über eine Unterbrechungskante den Unterbrechungsbereich, so werden alle Token und

damit auch alle Abläufe innerhalb des Unterbrechungsbereich gelöscht, auch solche, die von verschiedenen Aufrufen stammen.

Unterbrechungsbereich mit Unterbrechungskante



Ein Unterbrechungsbereich wird in Form eines abgerundeten Rechtecks mit strichlierten Linien dargestellt, die Unterbrechungskante wird, wie auch bei Ausnahmen üblich, entweder durch einen »Blitz« symbolisiert oder aber durch eine herkömmliche gerichtete Kante mit einem kleinen annotierten Blitzsymbol.

4.2.13 Aktionen

Aktionen als elementare Bau-steine beliebigen benutzer-definierten Verhaltens, das den Kontext darstellt und die Aktionen

Aktionen sind, wie bereits erwähnt, die elementaren Bausteine einer benutzerdefinierten Aktivität und dienen dem lesenden und schreibenden Zugriff auf Objekte, sowie zum Aufruf anderen Verhaltens. Aktionen können damit den Zustand des Systems verändern. Eine Aktion ist immer einer benutzerdefinierten Verhaltensbeschreibung - zum Beispiel einer Aktivität oder einer Transition in einem Zustandsdiagramm - zugeordnet und nur durch diese verfügbar. Die Verhaltensbeschreibung stellt den Kontext für eine Aktion dar und koordiniert diese indem festgelegt wird, wann eine Aktion mit welchen Eingabewerten ausgeführt wird. Damit lässt sich eine Analogie zu Programmiersprachen ziehen, wo sich eine Prozedur aus einer Reihe von elementaren Anweisungen zusammensetzt.

Aktionen sind atomar, können aber abgebrochen werden

Aktionen sind *atomar*, wenngleich eine *Unterbrechung* einer Aktion möglich ist. Atomarität bedeutet in diesem Fall, dass etwaige Effekte der Aktion rückgängig gemacht werden, so als wäre die Aktion nie gestartet worden.

Aktionen können Eingabewerte zu Ausgabewerten verarbeiten

Aktionen können *Eingabewerte* verarbeiten, die durch den Datenfluss über Eingabepins empfangen werden, und können nach deren Beendigung *Ausgabewerte* an den Ausgabepins zur Verfügung stellen. Werte werden dabei generell als Kopien zur Verfügung gestellt, Kopien von Objektreferenzen verweisen allerdings wiederum auf das Originalobjekt.

Ein Ausgabewert kann entweder sofort zur Verfügung gestellt werden, oder erst nachdem alle verfügbar sind

Zur Festlegung des Zeitpunkts, zu dem eine Aktion, die mehrere *Ausgabewerte* erzeugt, diese zur Weiterleitung zur Verfügung stellt, gibt es zwei Optionen (es handelt sich dabei um einen semantischen Variationspunkt - siehe Kapitel 5.1.6 ab Seite 336). Entweder werden alle Ausgabewerte einer Aktion gleichzeitig zur Verfügung gestellt sobald alle verfügbar sind oder jeder der Ausgabewerte wird sofort bei Verfügbarkeit zur Verfügung gestellt. In jedem Fall werden nach Beendigung einer Aktion neben den Ausgabewerten an allen ausgehenden Kontrollflusskanten der Aktion Kontrolltoken angeboten.

Vordefinierte Aktionen sind sprachunabhängig, atomares Verhalten kann jedoch auch in einer belie-bigen Programmiersprache definiert werden

In UML existieren eine Reihe *vordefinierter, sprachunabhängiger* Aktionen, die auf Grund ihrer Granularität einfach auf eine beliebige Zielsprache abgebildet werden können. Zusätzlich kann atomares Verhalten, sogenanntes *OpaqueBehavior,* in einer beliebigen Sprache definiert wer-

den. Beispiele dafür sind mathematische Funktionen (*FunctionBehavior*), die für Eingabewerte Ausgabewerte erzeugen, ansonsten aber nebeneffektfrei sind.

Spezielle Notation für bestimmte Aktionsarten selten

Für einen Grossteil der vordefinierten Aktionen gibt es keine eigenen Notationselemente, sie werden einfach textuell innerhalb des für Aktionen üblichen Symbols angeschrieben.

Kategorisierung der in UML vordefinierten Aktionen

Gemäss ihrer Funktion bzw. Komplexität können die in UML vordefinierten Aktionen in folgende Kategorien eingeteilt werden (siehe [Kath03]):

Primitive Aktionen

- Primitive Aktionen erlauben z.B. das Auslösen von Ausnahmen (siehe *Ausnahmebehandlung und Unterbrechungsbereich*, S. 227).

Kommunikationsbezogene Aktionen

- Kommunikationsbezogene Aktionen umfassen die Möglichkeit *Objekte* und *Signale* an *Empfängerobjekte* zu *übermitteln*, *Verhalten* und *Operationen aufzurufen* und schliesslich das Auftreten verschiedener Arten von *Ereignissen* zu *erkennen*.

Objektbezogene Aktionen

- Objektbezogene Aktionen umfassen Aktionen zum *Erzeugen* und *Löschen* von *Objekten*, zum *Aufruf* von *Objektverhalten*, zum Ermitteln aller *Objekte eines Classifiers* bzw. um deren *Zugehörigkeit* oder *Identität* zu testen, sowie um das für die Ausführung einer Aktion *zuständige Objekt* zu finden oder um ein Objekt zu *reklassifizieren*.

Strukturmerkmals- und variablenbezogene Aktionen

- Die Kategorie der strukturmerkmals- und variablenbezogenen Aktionen umfasst Aktionen zum *Setzen* und *Löschen* einzelner oder aller Werte von *strukturellen Merkmalen*, sowie von *Variablen*.

Linkbezogene Aktionen

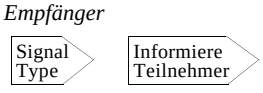
- Linkbezogene Aktionen ermöglichen das *Erzeugen* und *Löschen* bestimmter oder aller Links zwischen Objekten, sowie das *Navigieren* auf Basis von Links.

Im Folgenden werden die Aktionen, die diesen Kategorien angehören, näher erläutert, wobei auf eine detaillierte Behandlung von primitiven Aktionen verzichtet wird (dazu sei beispielhaft auf das Kapitel »*Ausnahmebehandlung und Unterbrechungsbereich*«, S. 227 verwiesen).

Kommunikationsbezogene Aktionen

SendObjectAction und SendSignalAction erlauben das Versenden von Objekten bzw. Signalen an bestimmte Empfänger

SendObjectAction und *SendSignalAction* erlauben ein Objekt oder ein Signal und die dazugehörigen Argumentwerte an ein Empfängerobjekt zu übermitteln. Es obliegt dem Empfänger, entsprechend zu reagieren, z.B. durch Anstoßen weiteren Verhaltens. Nach der Sendetätigkeit wird die Aktion beendet und mit den nachfolgenden Aktionen der Aktivität fortgesetzt. Die Kommunikation erfolgt asynchron, eventuelle Antworten des Empfängers werden vom Aufrufer ignoriert. Als Notationselement kann für beide Aktionsarten ein Blockpfeil verwendet werden.



BroadcastSignalAction erlaubt das Versenden von Signalen an beliebige Empfänger

Während eine *SendSignalAction* dazu dient, eine Signalinstanz zu erzeugen und an einen bestimmten Emfänger zu übermitteln, dient eine *BroadcastSignalAction* dazu, eine Signalinstanz an eine Reihe beliebiger Empfänger zu übermittlen. UML gibt dabei nicht vor, ob und wie die Menge der Signalempfänger eingeschränkt werden kann.

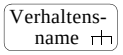
CallBehaviorAction und CallOperationAction ermöglichen den Aufruf von standalone-Verhalten bzw. von Operationen eines Objekts

CallBehaviorAction und *CallOperationAction* werden verwendet, um beliebiges Verhalten aufzurufen. Dies kann entweder im objektorientierten Sinn erfolgen, d.h. indirekt über den Aufruf der Operation eines Objekts (*CallOperationAction*) oder aber indem durch *CallBehaviorAction* unabhängig von einem Objekt, ein eigenständiges Verhalten aufgerufen wird. Die beiden Aufrufaktionen stellen demnach eine Indirektionsstufe dar, durch die die Definition einer Verhaltensbeschreibung von deren Verwendung explizit getrennt wird, was zu einer Erhöhung der Wiederverwendbarkeit führt.

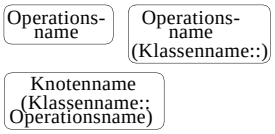
Ein- und Ausgabeparameter für das aufgerufene Verhalten, sowie synchroner und asynchroner Aufruf

Die vom eingebundenen Verhalten benötigten Parameter müssen durch die Eingabepins der Aktion zur Verfügung gestellt werden, wobei die Zuordnung implizit über die Reihenfolge von Pins und Parametern erfolgt. Ebenso werden Rückgabewerte, die nach Beendigung des aufgerufenen Verhaltens verfügbar sind, in den Ausgabepins der aufrufenden Aktion zur weiteren Verarbeitung bereitgestellt. Schliess-lich kann ein Aufruf entweder *synchron* mit Rückgabewerten oder *asynchron* und damit ohne Rückgabewerte erfolgen.

CallBehaviorAction



CallOperationAction



Als Notation für eine *CallBehaviorAction* wird innerhalb eines abgerundeten Rechtecks der Name des aufgerufenen Verhaltens eingetragen, sowie durch ein »Rechensymbol« in der rechten unteren Ecke der Aufruf eines anderen Verhaltens im Sinne einer Dekomposition symbolisiert. Für eine *CallOperationAction* gibt es drei verschiedene Notationsvarianten. Zum Einen kann der Operationsname direkt in das abgerundeten Rechteck hineingeschrieben werden, wobei optional in runden Klammern der entsprechende Klassenname, gefolgt von einem doppelten Doppelpunkt angegeben werden kann. Zum Anderen kann aber auch der Knoten, der die *CallOperationAction* repräsentiert, einen anderen Namen als die Operation aufweisen, wobei der eigentliche Operationsname hinter dem Klassennamen angegeben wird.

AcceptEventAction erkennt asynchrone Ereignisse und gibt diese zurück

Mit *AcceptEventAction* kann das Auftreten asynchroner Nachrichten und Aufrufe erkannt werden, z.B. ein Signal, das durch *SendSignalAction* generiert wurde. Der Typ der zu erkennenden Ereigniseintritte wird der Aktion über einen sogenannten »Trigger« zugeordnet. Prinzipiell werden Ereigniseintritte von jenen Objekten erkannt, die das entsprechende Verhalten besitzten und auch bei diesen gespeichert. Wird *AcceptEventAction* durch Anliegen entsprechender Token gestartet und liegt bereits ein passendes Ereignis vor, so wird eine Beschreibung des Ereignisses an den aus-

gehenden Kanten zurückgegeben, liegt kein passendes Ereignis vor, wird gewartet.

Asynchrones Ereignis

Asynchrones Zeitereignis

Als Notationselement für eine *AcceptEventAction* wird ein Rechteck mit einer Einkerbung verwendet, das keine eingehenden Kanten aufweisen darf. *AcceptEventAction* kann auch für absolute oder relative Zeitereignisse verwendet werden, wobei als Notation in diesem Fall eine Sanduhr verwendet wird.

AcceptCallAction zur Behandlung synchroner Ereignisse

AcceptCallAction ist eine spezielle *AcceptEventAction*, mit der Ereigniseintritte, die synchrone Operationsaufrufe repräsentieren, behandelt werden können. Die bei einem Aufruf durch *CallOperationAction* zur Verfügung gestellten Parameter sind als Ausgaben dieser Aktion verfügbar.

ReplyAction ermöglicht die Rückkehr zum Aurufer und die Beendigung des Aufrufs

ReplyAction ermöglicht es nach einer *AcceptCallAction* die Kontrolle wieder an den Aufrufer zurückzugeben und so die Ausführung des Aufrufs zu beenden. Als Eingabeparameter erhält die *ReplyAction* die »Rückkehrinformation«, die als Ausgabe der *AcceptCallAction* zur Verfügung steht. Die vom Aufrufer geforderten Rückgabewerte müssen als Eingabewerte an die *ReplyAction* übergeben werden.

Objektbezogene Aktionen

CreateObjectAction erzeugt ein Objekt eines Classifiers

Durch eine *CreateObjectAction* kann ein neues Objekt eines bestimmten *Classifiers* erzeugt werden. Dabei wird jedoch keine Initialisierung durchgeführt, sodass dieses Objekt weder Werte noch Beziehungen zu anderen Objekten aufweist. Der *Classifier* der als Eingabeparameter der Aktion mitgegeben wird, darf weder abstrakt, noch eine Assoziationsklasse sein. Das neue Objekt steht als Ausgabeparameter zur Verfügung.

DestroyObjectAction löscht ein Objekt eines Classifiers

Durch *DestroyObjectAction* kann ein Objekt das der Aktion als Eingabeparameter übergeben wird zur Laufzeit gelöscht werden. Ist das Objekt selbst ein Link, so wird automatisch auch eine *DestroyLinkAction* (siehe weiter unten) ausgeführt. Durch Kompositionsbeziehung verbundene Objekte können optional mitgelöscht werden (kann durch das Metaattribut *isDestroyOwnedObjects* festgelegt werden). Dasselbe gilt für Links die das Objekt zu anderen Objekten aufweist (*isDestroyLinks*).

StartClassifierBehaviorAction startet das Verhalten eines Objekts eines Classifiers

Das Verhalten eines Objekts kann explizit mit *StartClassifierBehaviorAction* gestartet werden. Die Aktion benötigt dazu ein Objekt als Einabeparameter, für dessen *Classifier* Verhalten definiert ist. Wurde das betreffende Verhalten bereits initiert, so hat diese Aktion keine Auswirkungen.

ReadIsClassifierObjectAction überprüft die Zugehörigkeit eines Objekts zu einem bestimmten

Alle Objekte eines *Classifiers* (seine »Extension«), können durch *ReadExtentAction* ermittelt werden.

Die Zugehörigkeit eines Objekts zu einem *Classifier* kann mit Hilfe der Aktion *ReadIsClassifierObjectAction* überprüft werden. Das zu überprü-

fende Objekt sowie der *Classifier* werden dabei der Aktion als Eingabeparameter mitgegeben, das Ergebnis ist ein boolscher Wert. Standardmässig wird geprüft, ob das Objekt direkte oder indirekte Instanz des *Classifiers* ist, was jedoch auf eine direkte Zugehörigkeit (*isDirect*) eingeschränkt werden kann.

TestIdentityAction überprüft zwei Objekte auf Identität

Durch *TestIdentityAction* kann geprüft werden, ob zwei Objekte ident sind.

ReadSelfAction erlaubt die Bestimmung des Objekts, das eine Aktion ausführt

Mit der Aktion *ReadSelfAction* kann eine Aktivität bestimmen, welches Objekt zur Laufzeit eine bestimmte Aktion ausführt. Dies setzt voraus, dass die Aktivität einem *Classifier* zugeordnet ist. Ist die Aktivität die Methode einer Operation, so darf diese nicht statisch sein.

ReclassifyObjectAction ermöglicht die Reklassifizierung von Objekten

Die Klassifizierung eines Objekts, d.h. dessen Zuordnung zu *Classifiern*, kann zur Laufzeit mit *ReclassifyObjectAction* geändert werden, wobei sowohl die Identität des Objekts als auch eventuell existierende Werte und Links gewahrt bleiben. Mit Hilfe dieser Aktion können dem Objekt ein oder mehrere neue *Classifier* hinzugefügt werden, wobei standardmässig die existierenden Klassifizierungen erhalten bleiben. Dieses Standardverhalten kann jedoch geändert werden (*isReplaceAll*). Ein neu hinzuzufügender *Classifier* darf nicht abstrakt sein und das Objekt muss letztendlich zumindest einem *Classifier* zugeordnet sein.

Strukturmerkmals- und variablenbezogene Aktionen

AddStructuralFeatureValue-Action und AddVariable-ValueAction erlauben das Setzen neuer Werte, bei Bedarf auf Basis von Positionsangaben

Die Aktionen *AddStructuralFeatureValueAction* erlaubt es einem strukturellen Merkmal, z.B. einem Attribut, neue Werte hinzuzufügen, wobei standardmässig alte Werte erhalten bleiben (änderbar durch *isReplaceAll*). Da ein strukturelles Merkmal *mehrwertig* und *geordnet* sein kann, stellt diese Aktion einen weiteren Eingabeparameter zur Verfügung, durch den in Form eines positiven Ganzzahlwertes die Position angegeben werden kann, an der ein neuer Wert eingefügt werden soll (*insertAt*). Wird durch die Aktion eine allfällige Multiplizitätseinschränkung des strukturellen Merkmals verletzt oder ist dieses als schreibgeschützt ausgewiesen, so ist die Semantik undefiniert. Analoges gilt für die Aktion *AddVariableValueAction* in Bezug auf Variablen.

ClearStructuralFeatureValue -Action und ClearVariableValueAction RemoveStructuralFeature ValueAction und RemoveVariableValueAction erlauben das Löschen bestimmter existierender Werte

Sollen alle Werte eines strukturellen Merkmals gelöscht werden, so kann dies in *einem* Schritt durch *ClearStructuralFeatureAction* erfolgen. Ein Unterschreiten der festgelegten Multiplizität wird dabei nicht überprüft. Ist das strukturelle Merkmal schreibgeschützt oder dürfen nur Werte hinzugefügt werden, so ist die Semantik dieser Aktion nicht definiert. Analoges gilt für die Aktion *ClearVariableValueAction* in Bezug auf Variablen.

Das gezielte Entfernen eines Werts eines strukturellen Merkmals oder einer Variablen ist durch *RemoveStructuralFeatureValueAction* bzw.

RemoveVariableValueAction möglich, wobei eine entsprechende Position angegeben werden muss (*removeAt*). Als weitere Option kann festgelegt werden, dass beim Löschen eines Werts auch alle Duplikate entfernt werden (*isRemoveDuplicates*).

Linkbezogene Aktionen

CreateLinkAction und CreateLinkObjectAction zur Erzeugung eines Links einer Assoziation ohne/mit Assoziationsklasse

Links zwischen zwei Objekten können durch *CreateLinkAction* erzeugt werden. Weist die zugehörige Assoziation eine Assoziationsklasse auf, muss *CreateLinkObjectAction* zur Erzeugung des entsprechenden Linkobjekts verwendet werden. Als Option kann für jede der zu erstellenden Enden des Links angegeben werden, ob bereits existierende Enden entfernt werden sollen. Bei geordneten Beziehungen kann eine Einfügeposition (*insertAt*) angeben werden.

DestroyLinkAction löscht bestimte Links und Linkobjekte

Eine *DestroyLinkAction* löscht Links zwischen Objekten, sowie etwaige Linkobjekte. Existieren Links zwischen gleichen Objekten mehrfach, so kann festgelegt werden, dass diese ebenfalls gelöscht werden (*isDestroyDuplicates*). Sind die Links geordnet, kann der zu löschende Link über eine Positionsangabe (*destroyAt*) spezifiziert werden.

ClearAssociationAction löscht alle Links einer Assoziation eines Objekts

Sollen alle Links einer Assoziation an denen ein Objekt teilnimmt gelöscht werden, so kann dies durch *ClearAssociationAction* erfolgen. Das Objekt, sowie die betreffende Assoziation repräsentieren die Eingabeparameter der Aktion.

ReadLinkAction zur Navigation von einem beliebigen Objekt zu einem Zielobjekt

Schliesslich erlauben drei verschiedene Aktionen das Navigieren entlang von Links, um ein entsprechendes Zielobjekt zurückzugeben. Bei *ReadLinkAction* kann von einem Ausgangsobjekt zu einem Zielobjekt navigiert werden, wobei die zugehörigen Assoziationsenden, sowie die Ausgangsobjekte als Eingabeparameter angegeben werden. Bei n-ären Beziehungen müssen alle beteiligten Objekte als Eingabeparameter übergeben werden, mit Ausnahme des Zielobjekts der Navigation.

ReadLinkObjectEndAction und ReadLinkObjectEndQualifierAction zur Navigation von einem Linkobjekt zu einem Zielobjekt ohne/mit Qualifikator

Bei *ReadLinkObjectEndAction* und *ReadLinkObjectEndQualifierAction* erfolgt die Navigation von einem Linkobjekt aus zu einem Zielobjekt ohne bzw. mit Nutzung eines entsprechenden Qualifikators.