Interaktionsdiagramme mit Datenspezifikation zur Darstellung verteilter Systeme*

Thomas Gehrke

Technische Universität Braunschweig
Institut für Software, Abteilung Programmierung
Gaußstraße 11, 38106 Braunschweig
gehrke@ips.cs.tu-bs.de

Zusammenfassung Wir definieren *n-Agenten-Diagramme* zur graphischen Spezifikation verteilter Systeme. Diese Diagramme basieren auf den Sequenzdiagrammen von UML, erweitern diese aber um die Darstellung von Datentransformationen und Gültigkeitsbereichen von Bezeichnern. Weiterhin enthalten sie spezielle Varianten von Auswahloperationen und Schleifen für die Verwendung in verteilten Systemen.

1 Einführung

Interaktionsdiagramme wie die Sequenzdiagramme aus UML [BRJ98,OMG99] und *Message Sequence Charts* [ITU96a,ITU99] werden häufig zur Spezifikation des Verhaltens von Systemen eingesetzt. Hierbei werden zwei Betrachtungsweisen von Diagrammen unterschieden: *Exemplarische* Diagramme zeigen einen möglichen Systemablauf, während *generische* Diagramme eine Menge von möglichen Systemabläufen definieren. Die Angabe generischer Diagramme wird durch komplexe Diagrammelemente wie Schleifen und Alternativen unterstützt.

Da die Interaktionsdiagramme kein konkretes Systemmodell vorgeben, können sowohl verteilte Systeme, in denen die Komponenten als parallel ablaufende Prozesse realisiert sind, als auch sequentielle prozedurale Systeme spezifiziert werden. Durch die Konzentration auf die Darstellung des Systemverhaltens abstrahieren Interaktionsdiagramme im allgemeinen von den während des Systemablaufs durchgeführten Datenberechnungen. Daher muß zur ergänzenden Spezifikation der Daten oft eine zusätzliche Darstellungsart herangezogen werden. Weiterhin sind die komplexen Sprachelemente wie Schleifen und Alternativen nur in einer sehr allgemeinen Weise angegeben. Zum einen abstrahieren diese Kontrollstrukturen vom zugrundeliegenden Systemmodell (z.B. verteilt oder prozedural), so daß keine Sprachelemente zur Zuordnung von Entscheidungen an Instanzen existieren. Dies erschwert die Angabe einer formalen Semantik für Diagramme. Zum anderen wird durch die fehlende Definition von Gültigkeitsbereichen nicht festgelegt, welche Elemente in den Ausdrücken der komplexen Kontrollstrukturen auftreten dürfen.

* Gefördert im Rahmen des Projekts Go 671/2-2 *Entwurf reaktiver Systeme* der Deutschen Forschungsgemeinschaft.

Im vorliegenden Papier führen wir eine neue Variante von Interaktionsdiagrammen ein, die wir *n-Agenten-Diagramme* nennen. Sie sind speziell für die Spezifikation verteilter Systeme vorgesehen, die aus parallelen Komponenten (genannt *Agenten*) bestehen und die durch das Versenden von Nachrichten interagieren. Wir nennen diese Systeme *n-Agenten-Systeme*. Die n-Agenten-Diagramme basieren auf den Sequenzdiagrammen von UML, erweitern diese aber um die Spezifikation von Daten. Die Syntax der UML-Sequenzdiagramme wurde aus Gründen der Kompatibilität zu existierenden Entwicklungswerkzeugen gewählt. Weiterhin sind die komplexen Sprachkonstrukte wie Alternativen und Schleifen an das Systemmodell der verteilten Komponenten angepaßt. Hierbei diskutieren wir die Nachteile der in Sequenzdiagrammen enthaltenen Konstrukte bzgl. der Verwendung für die Darstellung verteilter Systeme mit Daten.

In Abschnitt 2 erläutern wir das Modell der n-Agenten-Systeme und definieren die Sprachelemente, die wir aus den Sequenzdiagrammen übernehmen. Anschließend erweitern wir die Diagramme in Abschnitt 3 um die Beschreibung von Datenoperationen und Bindungen. In Abschnitt 4 definieren wir als komplexe Sprachelemente Alternativen und Schleifen. Abschnitt 5 enthält die Spezifikation eines kleinen Beispiels. Abschließend diskutieren wir in Abschnitt 6 die vorgestellten Ergebnisse und geben einen Ausblick auf weitere Forschungsarbeit.

2 Systemmodell

In diesem Abschnitt erläutern wir das Systemmodell der n-Agenten-Systeme und geben an, welche Elemente wir aus den Sequenzdiagrammen übernehmen.

Instanzen. Ein n-Agenten-System besteht aus einer Menge von n Komponenten, genannt *Instanzen*, die parallel ausgeführt werden. Die Instanzen verfügen über lokalen Speicher; ein globaler Speicher, auf den alle Instanzen zugreifen können, ist nicht vorgesehen. Der Austausch von Informationen zwischen den Instanzen findet durch Kommunikation statt. Hierzu verfügt jede Instanz über eine Menge von *Toren*, über die Nachrichten an die Instanz gesendet werden können. Diese Tore fungieren als Schnittstelle für die Interaktion mit der Instanz. Tore besitzen einen Typ, der angibt, welche Art von Parametern bei der Kommunikation über dieses Tor übertragen werden können. Wir geben Instanzen und die von ihnen bereitgestellten Tore in Form von Instanzen der UML-Klassendiagramme an. In Abbildung 1(a) ist die Deklaration einer Instanz I_1 angegeben, die die Tore m_1 und m_2 zur Interaktion mit ihrer Umwelt verwendet. Die Nachrichten an das Tor m_1 verwenden einen Parameter des Typs *int*. Das Tor m_2 ist ein Tor für Anfragen (s.u.). Beim Aufruf wird ein Parameter des Typs *int* übertragen. Ebenso enthält die Antwort an den Aufrufer einen Parameter des Typs *int*. Die in der Instanzdeklaration aufgeführten Tore sind zu Beginn des Systemablaufs global bekannt. Zusätzlich besteht die Möglichkeit, während des Systemablaufs weitere Tore zu erzeugen[1].

[1] Diese Eigenschaft ist z.B. für die Spezifikation von Sicherheitseigenschaften wünschenswert.

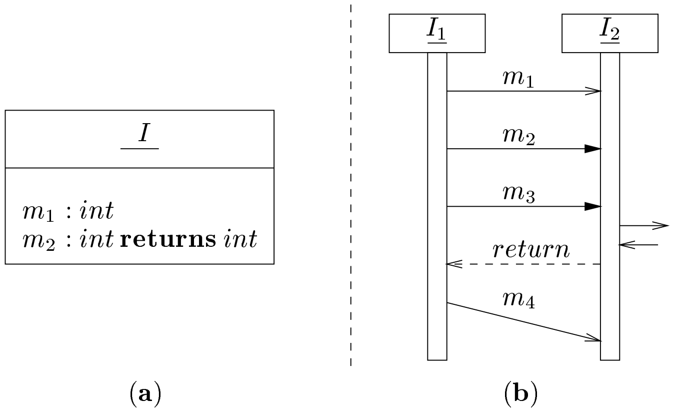


Abbildung 1. Instanzen und Nachrichten.

Kommunikation. Für die Darstellung der einzelnen Nachrichtenarten durch Pfeile verwenden wir die Notation von UML. Als Namen einer Nachricht verwenden wir das Tor, an das sie geschickt wird. Pfeile mit einer nicht ausgefüllten Spitze repräsentieren Nachrichten, bei denen Sender und Empfänger direkt nach dem Senden bzw. dem Empfangen der Nachricht in ihrer Ausführung fortfahren können (siehe die Nachricht m_1 in Abbildung 1(b)). Pfeile mit ausgefüllten Spitzen stellen Nachrichten dar, bei denen der Sender auf eine Antwort des Empfängers warten muß. Solche Paare aus Nachricht und Antwort nennen wir *Anfragen*. Führt der Empfänger zwischen dem Empfang der Nachricht und dem Senden der Antwort keine weiteren Aktionen durch, kann die Darstellung der Antwort entfallen (m_2 in Abbildung 1(b)). Andernfalls wird die Antwort als gestrichelter Pfeil dargestellt und erhält den Namen *return* (Nachricht m_3).

Nachrichten können synchron und asynchron ausgetauscht werden. Horizontale Pfeile kennzeichnen synchrone Nachrichten, während diagonale Pfeile asynchrone Nachrichten repräsentieren (siehe m_4 in Abbildung 1(b)). Bei synchronen Nachrichten müssen Sende- und Empfangereignis gemeinsam durchgeführt werden, während bei asynchronen Nachrichten der Sender nach dem Senden der Nachricht unabhängig vom Empfänger weiter ausgeführt werden kann. Bei asynchronen Anfragen muß auch die Antwort asynchroner Natur sein; wie bei synchronen Anfragen ist der Sender ist so lange blockiert, bis die Antwort eintrifft.

3 Beschreibung von Daten

Wir erweitern die in Abschnitt 2 eingeführten Diagramme um Elemente zur Beschreibung von Datentransformationen. Hierbei soll die Syntax von Sequenzdiagrammen möglichst nicht verändert werden, da die Einsetzbarkeit der Erwei-

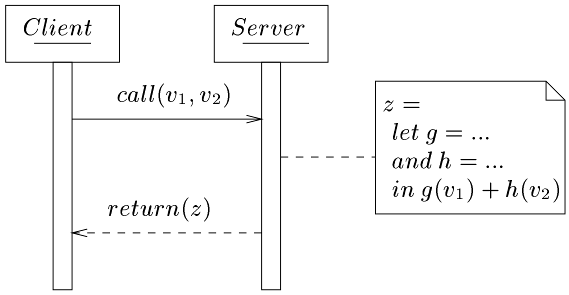


Abbildung 2. Datenbeschreibung mit Annotationen.

terungen von der Kompatibiltät zu existierenden Entwurfswerkzeugen abhängig ist.

Um eine möglichst problemorientierte Systembeschreibung zu ermöglichen, geben wir keine konkrete Datensprache an, sondern beschreiben generelle Ansätze zur Integration von Daten in Interaktionsdiagramme. Da auch die Integration nicht-imperativer Sprachen ermöglicht werden soll, verwenden wir im folgenden den Begriff *Bezeichner* statt *Variable.*

Annotationen. In UML ist vorgesehen, Elemente von Systemen, die nicht graphisch spezifiziert werden können, in Form von Annotationen (engl. *notes*) in den Diagrammen anzugeben. Annotationen sind graphische Elemente, die textuelle Informationen beinhalten. Typische Inhalte von Annotationen sind Kommentare, Bedingungen und Informationen bzgl. des Metamodells von UML.

In unserem Ansatz verwenden wir Annotationen zur Angabe von Datendeklarationen und -berechnungen. In Abbildung 2 ist ein Diagramm angegeben, in dem ein *Client* eine Anfrage an einen *Server* sendet. Der Aufruf, hier *call* genannt, enthält zwei Werte v_1 und v_2. Die Annotation im Server enthält einen Ausdruck in einer funktionalen Sprache, der v_1 und v_2 verwendet. Das Ergebnis der Berechnung wird an den Bezeichner z gebunden und anschließend mit der Antwort *return* an den *Client* gesendet.

Gültigkeit von Bezeichnern. Wir ordnen den in Diagrammen definierten Datenbezeichnern Gültigkeitsbereiche zu. In einem verteilten System ohne globalen Speicher ist die Verwendung globaler Bezeichner nicht sinnvoll, daher beschränken wir den Gültigkeitsbereich von Bezeichnern auf die Instanz, in denen sie definiert wurden. Ebenso dürfen die Parameter von Nachrichten nur die im jeweiligen Sender zum Zeitpunkt des Sendens bekannten Bezeichner enthalten.

Bindung empfangener Werte. Zur Angabe der Bezeichner, an die die Parameter einer empfangenen Nachricht gebunden werden sollen, kann eine Annotation an die Spitze des die Nachricht repräsentierenden Pfeils angefügt werden. Diese Annotation enthält nach dem Schlüsselwort **bind** eine Liste der Bezeichner, an

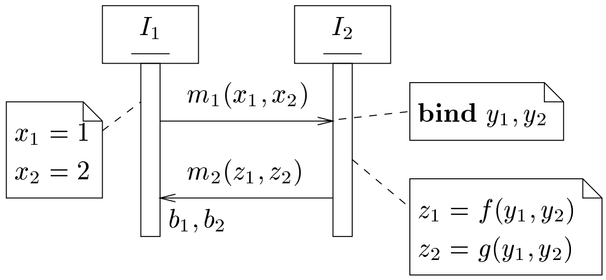


Abbildung 3. Bindung von gelesenen Werten an Bezeichner.

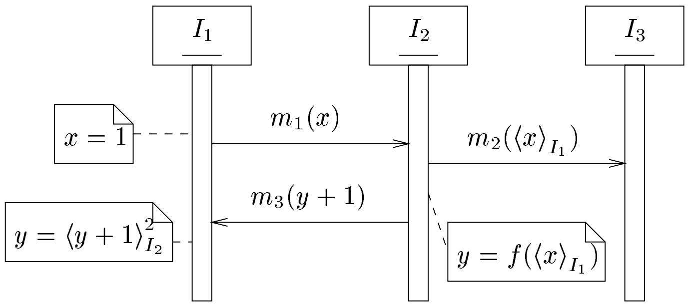


Abbildung 4. Verwendung impliziter Bezeichner.

die die Werte der Parameter gebunden werden sollen. In Abbildung 3 werden die von der Instanz I_1 gesendeten Werte von x_1 und x_2 in der Instanz I_2 an die Bezeichner y_1 bzw. y_2 gebunden, die dann in I_2 in Ausdrücken verwendet werden. Da das Binden gelesener Werte an Bezeichner in Diagrammen häufig vorkommt, erlauben wir eine abkürzende Schreibweise, bei der die Bezeichner ohne Annotation direkt an die Pfeilspitze geschrieben werden (siehe b_1, b_2 für die Nachricht m_2 in Abbildung 3). Eine explizite Typangabe für die Bezeichner ist nicht notwendig, da ihr Typ dem des jeweils zugehörigen Parameters entspricht.

Implizite Bezeichner. Für eine übersichtliche Darstellung des Datenflusses in einem System erlauben wir die Verwendung *impliziter Bezeichner.* Ein impliziter Bezeichner $\langle E \rangle_I$ repräsentiert den Wert des Ausdrucks E in Instanz I. In Abbildung 4 sendet I_1 den Wert von x an I_2. Diese sendet den empfangenen Wert an I_3 weiter. Anstelle einer expliziten Bindung an einen Bezeichner wird in I_2 der implizite Bezeichner $\langle x \rangle_{I_1}$ verwendet. Anschließend berechnet I_2 einen Wert für y und sendet den Wert $y + 1$ an I_1. Dort wird das Quadrat des Werts an den lokalen Bezeichner y gebunden. Die Verwendung der impliziten Bezeichner ermöglicht eine Darstellung des Übertragungswegs von Werten während des Systemablaufs, ohne daß in jeder beteiligten Instanz lokale Bezeichner verwen-

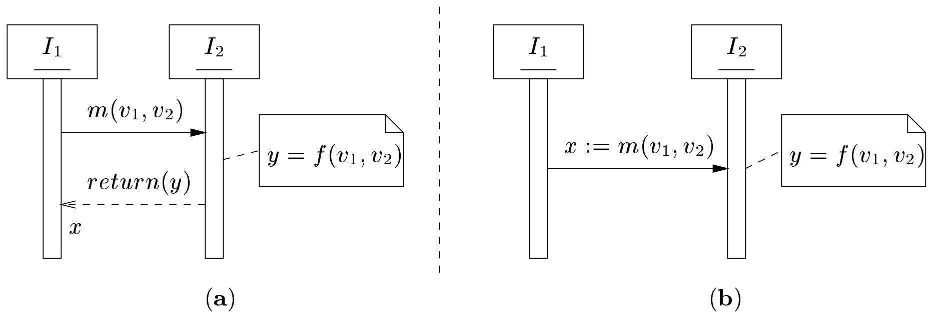


Abbildung 5. Abkürzende Notation für Anfragen.

det werden müssen. Ist die Zuordnung eines impliziten Bezeichners zu einer Instanz eindeutig, kann der Index I weggelassen werden (siehe Abbildung 12). Die Gültigkeitsregeln expliziter Bezeichner gelten analog für implizite Bezeichner; wir nehmen an, daß sie beim Empfangsereignis der entsprechenden Nachricht definiert werden (also $\langle x \rangle_{I_1}$ beim Empfang von m_1 bzw. m_2, und $\langle y+1 \rangle_{I_2}$ beim Empfang von m_3).

Anfragen. In Systemen mit parallelen Komponenten treten häufig Anfragen auf, bei denen die aufrufende Komponente nach dem Senden auf den Erhalt eines Ergebnisses wartet. Führt die Instanz, an die die Anfrage gesendet wurde, zwischen dem Empfang der Anfrage und dem Senden der Rückantwort nur Datenoperationen aus, kann für die Darstellung der Anfrage eine abkürzende Notation verwendet werden. In Abbildung 5(a) sendet I_1 eine Anfrage an I_2. I_2 berechnet aus den Parametern der Nachricht einen neuen Wert und sendet diesen als Ergebnis des Anfrage an I_1, wo er an den Bezeichner x gebunden wird. Da I_2 zwischen dem Empfang von m und dem Senden der Antwort keine Aktionen ausführt, kann die Anfrage, wie in Abbildung 5(b) angegeben, verkürzt dargestellt werden. Wir legen fest, daß in der abkürzenden Schreibweise die Werte aller Bezeichner, die seit dem Erhalt der Anfrage definiert wurden, in der Antwortnachricht als Parameter an den Aufrufer gesendet werden (in Abbildung 5 der Bezeichner y).

Dynamische Erzeugung von Toren. Wie bereits in Abschnitt 2 angeführt, erlauben wir die dynamische Erzeugung von Toren zur Laufzeit eines Systems. Die Erzeugung eines neuen Tores wird in einer Annotation dargestellt, die das Schlüsselwort **gate**, gefolgt von einer Liste aus Tornamen und zugehörigen Typen, enthält.

In Abbildung 6 wird in der Instanz I_1 dynamisch ein neues Tor m erzeugt, über das Nachrichten mit einem ganzzahligen Parameter gesendet werden können. Mit der Nachricht m_2 wird das neue Tor der Instanz I_2 bekanntgemacht (der Name wird als Wert übertragen), die dann den Tornamen an I_3 weitersendet.

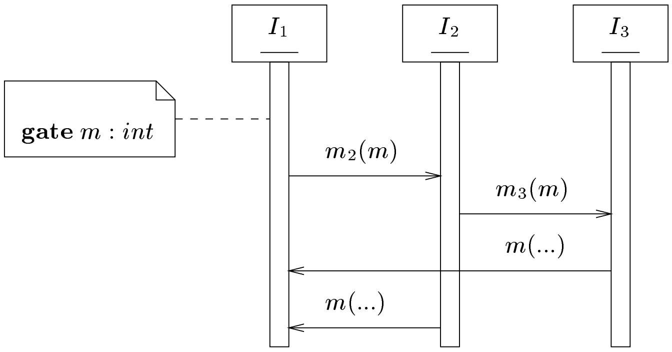


Abbildung 6. Dynamische Erzeugung von Tornamen.

Anschließend senden sowohl I_2 als auch I_3 Nachrichten über m an I_1. Obwohl Tornamen als Werte verschickt werden, können sie nicht wie Werte anderer Datentypen in Datenstrukturen abgelegt oder an Bezeichner gebunden werden. Andernfalls müssten wir anstelle des Tornamens in einer Nachricht einen Ausdrucke zulassen, der den zu verwendenden Tornamen berechnet. Dies würde zu einer Reduzierung der Lesbarkeit der Diagramme führen. Daher nehmen wir an, daß empfangene Tornamen direkt ohne die Bindung an Bezeichner verwendet werden können.

Um Tore, die Nachrichten mit Toren als Parameter erhalten können, zu typisieren, verwenden wir den Typkonstruktor *gate*. Die Tore m_2 und m_3 in Abbildung 6 besitzen somit den Type (*int gate*) *gate*. Während das Recht, Nachrichten an ein Tor zu senden, durch die Weitergabe des Tornamens an andere Instanzen übertragen werden kann, ist das Empfangen von Informationen über ein Tor nur der Instanz möglich, die das Tor erzeugt hat. Daher können I_2 und I_3 in Abbildung 6 keine Nachrichten über m empfangen.

4 Komplexe Sprachelemente

In diesem Abschnitt führen wir als komplexe Sprachelemente Alternativen und Schleifen ein. Diese Konstrukte unterscheiden sich von den entsprechenden Elementen in [BRJ98,OMG99], da sie besser an das in Abschnitt 2 vorgestellte Systemmodell und Datenbehandlung angepaßt sind, während die Konstrukte in UML eher allgemeiner Natur sind.

Auswahl zwischen Alternativen. In UML werden Alternativen durch das bedingte Versenden von Nachrichten realisiert. In Abbildung 7 sendet I_1 die Nachricht m nur an I_2, wenn die Bedingung $x > 0$ erfüllt ist. Dieses Konzept der bedingten Nachrichten ist für parallele Systeme nur bedingt geeignet (siehe auch [GGW99]). Zum einen muß I_2 als aktiver Prozeß auch dann fortfahren können, wenn die Nachricht m nicht gesendet wird. Daher muß I_2 entweder wissen, ob

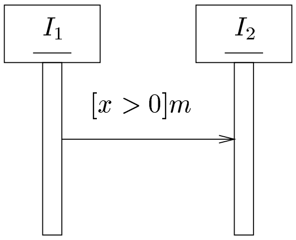


Abbildung 7. Sequenzdiagramm mit Bedingung.

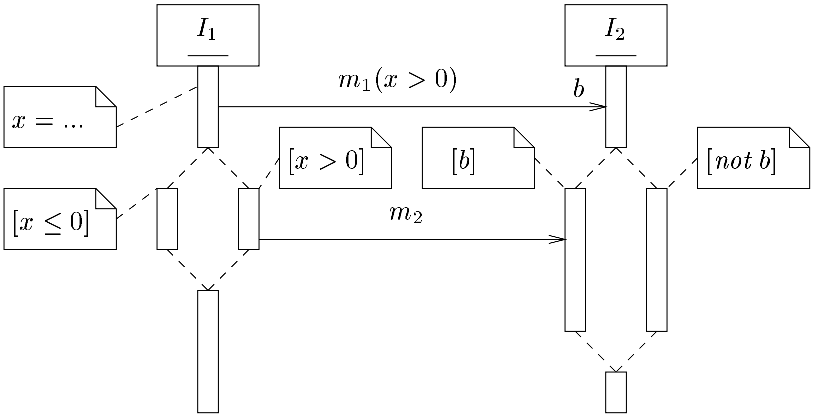


Abbildung 8. n-Agenten-Diagramm mit Auswahloperatoren.

m gesendet wird, oder es muß eine Zeitbeschränkung (*time out*) für m gesetzt werden. I_1 und I_2 besitzen keinen gemeinsamen Speicher, daher kann I_2 nicht auf den Wert von x in I_1 zugreifen, um das Resultat von $x > 0$ zu ermitteln. Wird eine Zeitbeschränkung für das Warten auf m eingeführt und die Nachricht trifft nicht ein, können die Bezeichner, an die die Parameter von m gebunden werden sollten, nicht mit Werten belegt werden. Daher können in I_2 offene Ausdrücke entstehen, die nicht auswertbar sind. Eine mögliche Abhilfe besteht in der Angabe von *default*-Werten für die Bezeichner, was aber von vielen Programmiersprachen nicht unterstützt wird.

Wir definieren eine eigene Notation zur Darstellung von alternativem Verhalten, bei der die Auswahl der Alternative lokal innerhalb einer Instanz getroffen wird. Syntaktisch wird eine Auswahl durch eine Aufspaltung der Lebenslinie dargestellt. Die Bedingungen der Alternativen werden in Annotationen in der gewohnten UML-Notation in eckigen Klammern angegeben. Aus Gründen der Übersichtlichkeit kann die Bedingung auch direkt ohne den Annotationsrahmen an den Beginn der Lebenslinie der Alternative geschrieben werden. Gelten in ei-

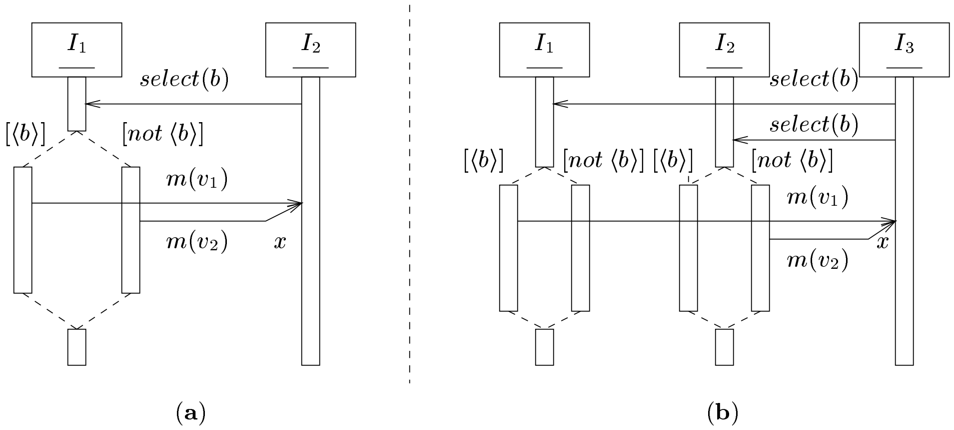


Abbildung 9. Diagramm mit alternativen Eingabeaktionen.

ner Auswahl mehr als eine Bedingung, wird eine Alternative nichtdeterministisch ausgewählt.

In Abbildung 8 wird in beiden Instanzen eine Auswahl zwischen zwei Alternativen getroffen. Zuerst wird in I_1 ein Bezeichner x definiert, an den das Ergebnis einer Berechnung gebunden wird. Das Resultat der Bedingung $x > 0$ wird mit der Nachricht m_1 an I_2 übermittelt, wo es an den Bezeichner b gebunden wird. Gilt $x > 0$, wird eine Nachricht m_2 an I_2 gesendet, andernfalls findet keine weitere Kommunikation statt.

Der Gültigkeitsbereich von Bezeichnern, die innerhalb einer Alternative definiert werden, erstreckt sich nur bis zum Ende der Alternative. Würde er über das Ende der Auswahl hinausreichen, müßte gewährleistet sein, daß in allen Alternativen der Auswahl dieselbe Menge von Bezeichnern definiert wird, da andernfalls offene Ausdrücke entstehen können. Ebenso muß die Antwort auf eine in einer Alternative empfangenen Anfrage innerhalb der Alternative gesendet werden.

Alternative Eingabe. Wir führen als weiteres Konstrukt die Auswahl zwischen mehreren Möglichkeiten zum Empfang einer Nachricht ein. In Abbildung 9(a) trifft I_1 eine Auswahl zwischen zwei Alternativen. In jeder Alternative sendet I_1 die Nachricht m an I_2, wobei unterschiedliche Werte übertragen werden. Den alternativen Empfang von $m(v_1)$ bzw. $m(v_2)$ stellen wir durch einen gemeinsamen Endpunkt der beiden Pfeile dar. Je nach Wahl der Alternative in I_1 erhält x in I_2 den Wert v_1 bzw. v_2. Alternative Instanzen einer Nachricht können auch von verschiedenen Instanzen gesendet werden. In Abbildung 9(b) wird eine Interaktion spezifiziert, bei der I_3 in Abhängigkeit des booleschen Werts von b entweder einen Wert von I_1 oder von I_2 anfordert. Alternative Nachrichten müssen dasselbe Tor verwenden, müssen einheitlich synchron bzw. asynchron übertragen werden, und es muß gewährleistet sein, daß genau eine der möglichen Nachrich-

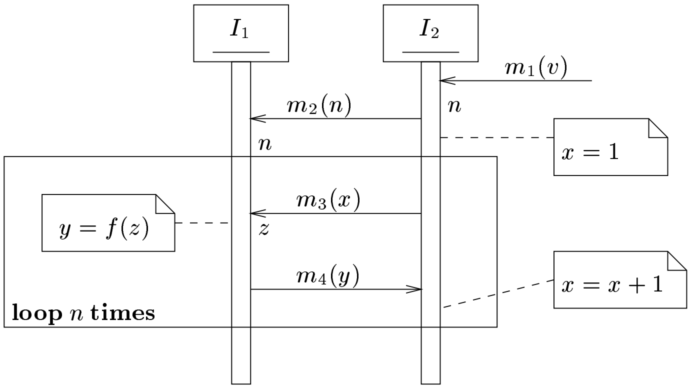


Abbildung 10. *loop*-Schleifen.

ten gesendet wird. Wir erlauben nicht die Spezifikation alternativer Anfragen, da dies die Modellierung alternativer Antworten erfordern würde. In diesem Fall würde bei den möglichen Empfängern der Antworten dieselben semantischen Probleme wie in Abbildung 7 auftreten.

Schleifen. In UML wird die Verwendung von Schleifen nicht explizit behandelt; in [OMG99] wird nur angegeben, daß die wiederholte Ausführung von Teilen einer Interaktion durch das Einrahmen der entsprechenden Nachrichten mit einem Kasten dargestellt werden soll. Am Boden dieses Kastens soll dann die Schleifenbedingung angegeben werden.

Bei der Verwendung des allgemeinen Schleifenkonstrukts für verteilte Systeme treten ähnliche Probleme wie bei Alternativen auf. Da die Schleifenbedingung global für alle an der Schleife beteiligten Instanzen definiert wird, ist nicht klar, wie die einzelnen Instanzen die Bedingung ohne die Existenz eines gemeinsamen globalen Speichers überprüfen können.

Wir führen zwei Arten von Schleifen ein. Der erste Schleifentyp ist eine *loop*-Schleife, bei der die Anzahl der Schleifendurchläufe vor Beginn der ersten Iteration bekannt ist. Zudem muß dieser Wert in allen beteiligten Instanzen vor Eintritt in die Schleife an denselben Bezeichner gebunden sein. Da die Anzahl der Iterationen bekannt ist, kann dann jede Instanz lokal die Durchführung der Schleife steuern. In Abbildung 10 ist eine *loop*-Schleife angegeben. Die Instanz I_2 erhält aus der Systemumgebung die Anzahl der Iterationen, bindet diesen Wert an n und übermittelt ihn an I_1, wo er ebenfalls an n gebunden wird. Die Schleife wird nun n-mal durchlaufen.

Der zweite Schleifentyp ist die *while*-Schleife. Bei diesem Schleifentyp wird eine Instanz als Kontrollinstanz ausgewählt. Diese spezielle Instanz überwacht die Gültigkeit der Schleifenbedingung, indem sie den entsprechenden Ausdruck lokal auswertet. Das Resultat der Auswertung wird dann durch *implizite Nachrichten* den anderen Instanzen mitgeteilt, so daß diese über die Durchführung

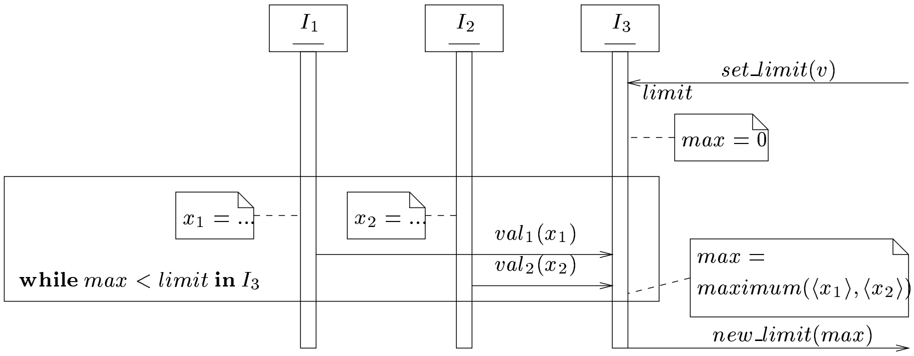


Abbildung 11. *while*-Schleife.

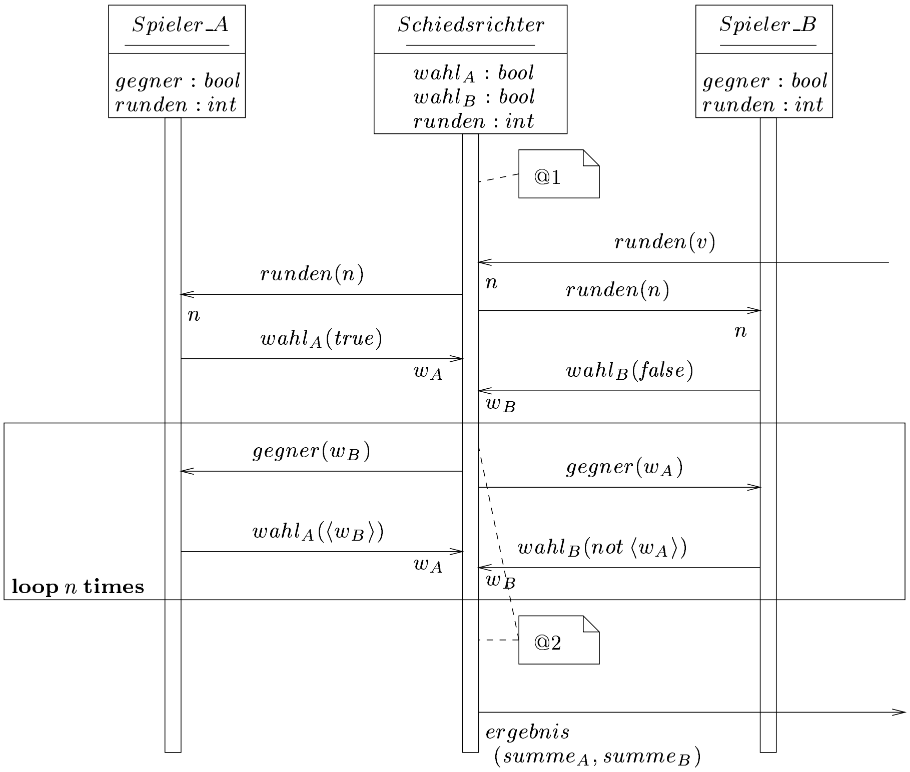
einer weiteren Iteration informiert sind. Die impliziten Nachrichten und die zugehörigen Tore werden nicht explizit im Diagramm graphisch dargestellt, um die Lesbarkeit nicht zu beeinträchtigen.

Abbildung 11 enthält ein Diagramm mit *while*-Schleife, bei der I_3 als Kontrollinstanz fungiert. Die Systemumgebung sendet zu Beginn der Ausführung einen Wert, der als Limit verwendet wird. In jeder Iteration werden von I_1 und I_2 Werte an I_3 gesendet. Ist das Maximum der beiden Werte kleiner als das Limit, wird die Schleife erneut ausgeführt; andernfalls wird das Maximum als neues Limit an die Umgebung gesendet.

Beide Schleifentypen überprüfen die Gültigkeit der Schleifenbedingung zu Beginn der Schleife; die Angabe der Bedingung am Boden des Schleifenrahmens wurde nur aus Kompatibilität zu [OMG99] gewählt. Nach Ausführung einer Schleife gelten alle Bezeichner, die in der letzten Schleifeniteration bekannt waren, also auch alle Bezeichner, die in der Schleife definiert wurden.

5 Beispiel

Als Beispiel für die Darstellung eines Systems spezifizieren wir das aus der Spieltheorie bekannte *Gefangenendilemma* (z.B. in [Hof88]). An diesem Spiel sind zwei Spieler beteiligt. Zu Beginn wird eine feste Anzahl von Spielrunden festgelegt. In jeder Runde können die Spieler aus zwei verschiedenen Alternativen wählen: Ein Spieler kann mit seinem Gegner kooperieren oder ihn betrügen. Entscheiden sich beide Spiele für Kooperation, erhalten beide 3 Punkte. Betrügt ein Spieler und der andere kooperiert, erhält der Betrüger 5 Punkte, während sein Gegner keinen Punkt erhält. Betrügen beide, erhalten sie jeweils einen Punkt. Die Spieler müssen nun versuchen, durch eine möglichst optimale Strategie die größte Anzahl von Punkten zu erhalten. Das Spiel wird durch einen Schiedsrichter koordiniert, der die Züge der Spieler entgegennimmt und die Punktekonten verwaltet.



@1	auswerten(x:$bool$,y:$bool$) = if x and y then return 3 elsif x and not y then return 5 elsif not x and y then return 0 else return 1 $summe_A = 0$ $summe_B = 0$
@2	$summe_A = summe_A$+auswerten(w_A, w_B) $summe_B = summe_B$+auswerten(w_B, w_A)

Abbildung 12. Beispiel: Gefangenendilemma.

Das Spiel ist in Abbildung 12 als Diagramm dargestellt. Die Systemumgebung sendet anfänglich die Anzahl der zu spielenden Runden an den Schiedsrichter, der diese an die Spieler weiterleitet. Dann machen die Spieler ihren ersten Zug. In der anschließenden Schleife wird die jeweils vorherige Spielrunde ausgewertet und die Punktekonten der Spieler aktualisiert. Dann wird den Spielern der vorherige Zug des jeweiligen Gegners mitgeteilt. Anschließend erhält der Schiedsrichter die neuen Züge der Spieler. Nach Beendigung der Schleife wird der letzte Zug ausgewertet und das Ergebnis an die Umgebung gesendet. *Spieler_A* spielt die Strategie *Tit for Tat* [Hof88], d.h., er kooperiert in der ersten Spielrunde und führt in jeder weiteren Runde den vorhergehenden Zug des Gegners aus. *Spieler_B* betrügt in der ersten Runde und führt in jeder weiteren Runde das Gegenteil vom vorherigen Zug des Gegners aus.

Die Datentransformationen geben wir in einer abstrakten Programmiersprache an. Aus Gründen der Übersichtlichkeit tragen wir in Abbildung 12 in die Annotationen nur Verweise auf ein *Datenverzeichnis* ein, das die Datentransformationen tabellarisch zusammenfaßt (vergleichbar dem *data dictionary* aus [DeM79]). Hierbei bezieht sich @i auf den i-ten Eintrag des Verzeichnisses.

6 Diskussion

In diesem Artikel haben wir eine graphische Notation für die Darstellung des Verhaltens und der Datentransformationen verteilter Systeme vorgestellt. Die Notation basiert auf den Konzepten der Sequenzdiagramme von UML, erweitert diese aber um die Beschreibung von Daten und Bindungen. Weiterhin enthält die Notation alternative Konstrukte zur Darstellung von Auswahloperatoren und Schleifen, die mehr als die allgemeinen Konstrukte in UML an die Spezifikation verteilter Systeme angepaßt sind. Dabei legen wir keine konkrete Datensprache fest, sondern erlauben die Verwendung beliebiger Datensprachen. Die Verwendung einer graphischen Notation erlaubt eine übersichtliche Darstellung des Systemverhaltens. Zudem ermöglicht die kombinierte Spezifikation von Kontroll- und Datenfluß die umfassende Beschreibung von Systemen innerhalb eines Diagrammtyps.

Message Sequence Charts. In den neuen Standard der *Message Sequence Charts* (MSC) [ITU99] wurde ebenfalls die Spezifikation von Daten aufgenommen. In Abbildung 13 ist ein MSC angegeben, in dem die Instanz I_1 eine Nachricht m_1 mit einem Parameter v_1 an I_2 sendet. Dort wird der empfangene Wert der Variablen x zugewiesen. Anschließend wird in einer internen Aktion eine Funktion auf x angewendet und das Resultat z zugewiesen. Dieser Wert wird an I_1 gesendet und dort an den Bezeichner y gebunden.

Wie in unserem Ansatz wird keine konkrete Datensprache festgelegt. Ebenso sind Bezeichner nur in der Instanz bekannt, in der sie definiert wurden. Allerdings werden in [ITU99] keine weiteren Informationen über die Bindung von Bezeichnern in bezug auf Schleifen oder Alternativen angegeben. Weiterhin unterstützen MSC auch nicht die dynamische Erzeugung von Toren.

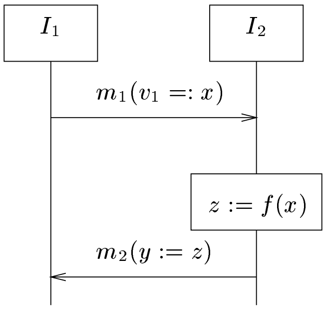


Abbildung 13. Message Sequence Chart mit Daten.

Semantik. Analog zur textuellen Darstellung von MSC wurde für die n-Agenten-Diagramme eine textuelle Notation entwickelt. Diese vereinfacht die Definition einer formalen Semantik für die Diagramme, da die Darstellung der Diagramme in textueller Form die Angabe von Übersetzungsfunktionen unterstützt. Wir haben eine Semantik definiert, die die textuelle Form der Diagramme in Terme einer Prozeßalgebra [Hoa85,Mil89,Fok00] übersetzt. Da die Semantik sowohl die Behandlung von Daten als auch das dynamische Erzeugen von Toren modellieren können muß, wurde die Prozeßalgebra $\mathcal{P}$ aus [GR99] verwendet, die das aus dem π-Kalkül [MPW92] bekannte Konzept der Mobilität mit der Beschreibung von Daten verbindet. Die bisherigen Ansätze zur Definition einer Semantik für Interaktionsdiagramme auf der Basis von Prozeßalgebren [ITU96b,MR97,GHRW98] konzentrieren sich auf das in den Diagrammen beschriebene Verhalten und erlauben keine Modellierung der Daten.

Tore werden durch Kommunikationskanäle realisiert. Das Senden eines Datentupels $v_1, ..., v_n$ an ein Tor m wird durch die Sendeaktion $m!v_1, ..., v_n$ auf dem gleichnamigen Kanal dargestellt. Das zugehörige Empfangen von Werten wird durch den Term

$$m?\_val; \{x_1 \leftarrow \#1\ \_val, ..., x_n \leftarrow \#n\ \_val\}; t$$

realisiert: Der Tupel der Parameter wird an den Hilfsbezeichner *_val* gebunden; anschließend werden die Komponenten des Tupels an die nach **bind** angegebenen Bezeichner $x_1, ..., x_n$ gebunden, die dann im Term t, der die Übersetzung der nachfolgenden Lebenslinie des Empfängers darstellt, definiert sind[2]. Anfragen werden definiert, indem im Sender ein neuer Kanal für die Antwort erzeugt und als zusätzliches Element im Parametertupel übertragen wird. Der Empfänger verwendet dann diesen Kanal zum Übertragen der Antwortdaten. Alternativen werden durch Auswahloperatoren modelliert, Schleifen durch rekursive Prozeßdefinitionen. Die Definition der textuellen Notation und der Semantik wird in [Geh00] angegeben.

[2] Die Funktion $\#i$ liefert den Wert der i-ten Komponente eines Tupels.

Ausblick. Gegenstand weiterer Forschung ist die Erweiterung der n-Agenten-Diagramme um die aus den *High-level Message Sequence Charts* bekannten Mechanismen zur Komposition von einzelen Diagrammen zu einer umfassenden Systembeschreibung [MR97]. Hierzu muß untersucht werden, in welcher Weise die Kompositionsoperatoren um die Beschreibung von Daten erweitert werden können. Ebenso planen wir die Analyse der Dekomposition von einzelnen Instanzen in ihre Komponenten unter Einbeziehung von Daten.

Literatur

[BRJ98] Grady Booch, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[DeM79] Tom DeMarco. *Structured Analysis and Systems Specification.* Prentice Hall, 1979.

[Fok00] Wan Fokkink. *Introduction to Process Algebra.* Springer, 2000.

[Geh00] Thomas Gehrke. *Dynamische Modelle für reaktive Systeme mit Daten.* Dissertation, Technische Universität Braunschweig, Institut für Software, Abteilung Programmierung, 2000.

[GGW99] Thomas Gehrke, Ursula Goltz, Heike Wehrheim. Zur semantischen Analyse der dynamischen Modelle von UML mit Petri-Netzen. In E. Schnieder (Hrsg.), *Tagungsband der 6. Fachtagung "Entwicklung und Betrieb komplexer Automatisierungssysteme"*, S. 547–566. Braunschweig, 1999.

[GHRW98] Thomas Gehrke, Michaela Huhn, Arend Rensink, Heike Wehrheim. An Algebraic Semantics for Message Sequence Chart Documents. In Stan Budkowski, Ana Cavalli, Elie Najm (Hrsg.), *Proceedings of FORTE/PSTV '98*, S. 3–18. Kluwer Academic Press, 1998.

[GR99] Thomas Gehrke, Arend Rensink. A Mobile Calculus with Data. Technischer Bericht 99-04, Technische Universtität Braunschweig, Institut für Software, Abteilung Programmierung, Oktober 1999. Erhältlich unter `http://www.cs.tu-bs.de/ips/gehrke/publications.html`.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Hof88] Douglas R. Hofstadter. *Metamagicum – Fragen nach der Essenz von Geist und Struktur.* Klett–Cotta, 1988.

[ITU96a] International Telecommunication Union ITU. Message Sequence Chart (MSC). Standard ITU-T Z.120, 1996.

[ITU96b] International Telecommunication Union ITU. Message Sequence Chart (MSC). Standard ITU-T Z.120 (Annex B), 1996.

[ITU99] International Telecommunication Union ITU. MSC 2000. Standard ITU-T Z.120, November 1999.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[MPW92] Robin Milner, Joachim Parrow, David Walker. A Calculus of Mobile Processes, Part I+II. *Information and Computation*, 100, 1992.

[MR97] S. Mauw, M.A. Reniers. High-level Message Sequence Charts. In *Proceedings of SDL '97:Time for Testing – SDL, MSC and Trends.* Elsevier, 1997.

[OMG99] Object Management Group OMG. Unified Modeling Language Specification. Technischer Bericht, Version 1.3 R9, 1999.