



Hauptseminar Semantik der UML 2.0

UML 2.0 Interaktionen
Semantik und Verfeinerung

Betreuer: Peter Graubmann

Bearbeiter: Fei Xie

INHALTSVERZEICHNIS

1 EINLEITUNG.....	3
2 GESCHICHTE UND DIAGRAMMARTEN DER UML 2.0.....	3
3 SEQUENZDIAGRAMME.....	5
3.1 Basis-Sequenzdiagramme.....	5
3.2 Strukturierte Sequenzdiagramme.....	7
4. SEMANTIK.....	11
4.1 Die Präliminarien für die Semantik der Interaktionen.....	11
4.2 Die Semantik des positiven Fragments.....	12
4.3 Die Negationen.....	13
4.4 Die Semantik das negativen Fragments.....	14
5. VERBESSERUNG DER NEGATION.....	15
5.1 σ -Funktionen.....	15
5.2 ν -Funktionen.....	16
6. IMPLEMENTIERUNG UND VERFEINERUNG	17
7. ZUSAMMENFASSUNG.....	18
8. LITERATURVERZEICHNIS.....	19

1. Einleitung

Diese Seminararbeit basiert auf [1]: "UML 2.0 Interactions: Semantics and Refinement" (CSDUML'04 Proceedings) 2004 von Maria Victoria Cengarle (Technische Universität) und Alexander Knapp (Ludwig-Maximilians-Universität München).

Die in UML 2.0 beschriebenen Semantiken für Interaktionsdiagramme, besonders für die Negation, sind grob und undeutlich. Das Ziel der Autoren war, eine formale Grundlage für eine eindeutige Definition des Negationsoperators der Interaktionsdiagramme in UML 2.0 zur Verfügung zu stellen.

Die Kernsprache, die in dieser Arbeit benutzt wurde, ist die Sprache für die verneinte Interaktion. Es hat sich herausgestellt, dass bei der Definition der formalen Semantik für diese Sprache viele Fragen entstanden. Diese betrafen zum Beispiel die klare Klassifizierung der verneinten Interaktionen. In diesem Dokument werden diese Probleme identifiziert und versucht, durch klassische Negation zu lösen.

Für weitere Informationen zu Grundlagen der UML können [6]: Object Management Group. Unified Modeling Language Specification, Version 2.0 (Superstructure). Adopted draft, OMG 2003, und die im Literaturverzeichnis angegebenen Veröffentlichungen benutzt werden.

2. Geschichte und Diagrammarten der UML 2.0

Die Unified Modeling Language ("vereinheitlichte Modellierungssprache"), üblicherweise mit UML abgekürzt, ist eine von der Object Management Group (OMG) entwickelte und standardisierte Beschreibungssprache, um Strukturen und Abläufe in objektorientierten Softwaresystemen darzustellen. In diesem Sinne einer Sprache definiert die UML dabei einen Bezeichner für die meisten Begriffe, die im Rahmen der Objektorientierung entstanden sind, und legt mögliche Beziehungen zwischen diesen Begriffen fest. Die UML definiert darüber hinaus grafischen Notationen für diese Begriffe und für Modelle von Software oder anderen Abläufen, die man in diesen Begriffen formulieren kann. Die Väter der UML (insbesondere Grady Booch, Ivar Jacobson und James Rumbaugh, auch "die drei Amigos" genannt) waren in den 90er-Jahren bekannte Propagandisten der objektorientierten Programmierung, die alle bereits ihre (mehr oder weniger ähnlichen) eigenen Systeme entwickelt hatten.

Als sie zusammen beim Unternehmen Rational Software beschäftigt waren, entstand der Ansatz, die verschiedenen Notationssysteme strukturiert zusammenzuführen. Die UML wurde am 19. November 1997 von der OMG als Standard akzeptiert und wird seitdem von ihr weiter entwickelt. Im Juni 2003 wurde die jüngste Version 2.0 der UML von der OMG als Entwurf veröffentlicht. Die UML 2.0 wurde im November 2004 endgültig verabschiedet [13].

UML 2.0 unterstützt insgesamt 14 Diagrammarten. Sie werden in zwei Gruppen, Strukturdiagramme und Verhaltensdiagramme, unterteilt. Strukturdiagramme haben folgende Diagrammarten: Klassendiagramm, Objektdiagramm, Paketdiagramm, Komponentendiagramm, Verteilungsdiagramm und Kompositionsstrukturdiagramm. Verhaltensdiagramme können eine von acht Formen haben: Interaktionsübersichtsdiagramm, Kommunikationsdiagramm, Aktivitätsdiagramm, Sequenzdiagramm, Anwendungsfalldiagramm, Zustandsdiagramm, Kollaborationsdiagramm oder Zeitverlaufdiagramm. (siehe Abb. 1).

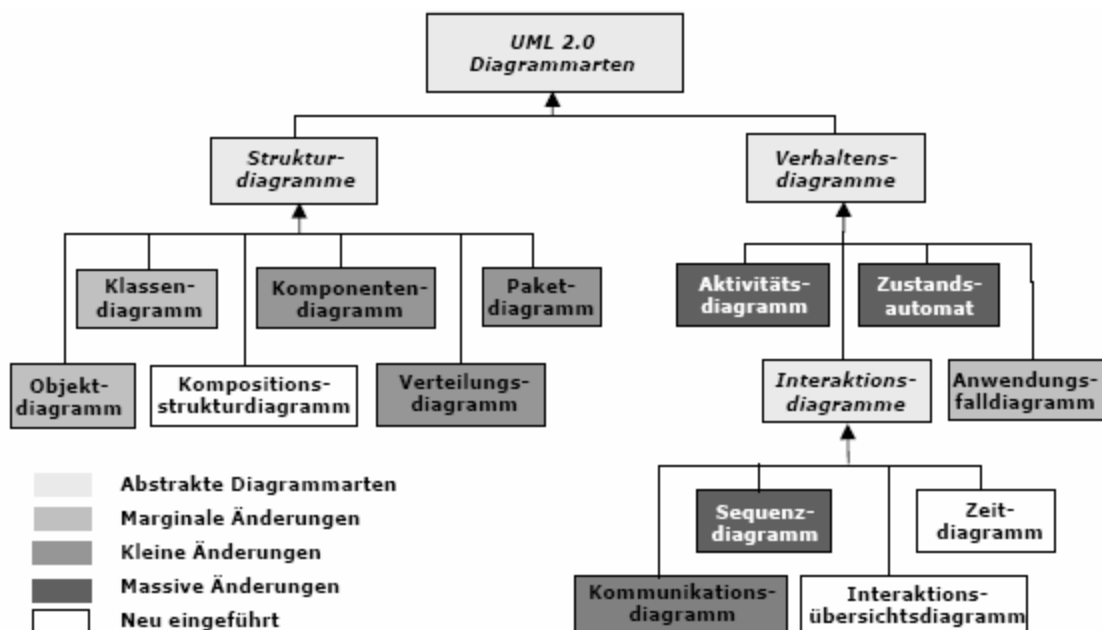


Abbildung 1. Diagrammarten der UML 2.0 [14]

Aus Abbildung 1 ersehen wir, dass es vier Arten von Interaktionsdiagrammen gibt: Kommunikationsdiagramme, Sequenzdiagramme, Interaktionsübersichtsdiagramme und Zeitdiagramme. Interaktionsdiagramme beschreiben die Wechselwirkungen zwischen einer Anzahl von Objekten im System. In dieser Arbeit werden wir uns auf Sequenzdiagramme konzentrieren.

3. Sequenzdiagramme

Sequenzdiagramme sind die wichtigsten Interaktionsdiagramme. Ein Sequenzdiagramm beschreibt die zeitliche Abfolge von Interaktionen zwischen einer Menge von Objekten innerhalb eines zeitlich begrenzten Kontextes. Mittels des Sequenzdiagramms beschreibt man die Interaktionen zwischen den Modellelementen ähnlich, wie bei einem Kommunikationsdiagramm, jedoch steht beim Sequenzdiagramm der zeitliche Verlauf des Nachrichtenaustausches im Vordergrund.

3.1 Basis-Sequenzdiagramme

Die Objekte werden lediglich mit senkrechten Lebenslinien dargestellt. Die Lebenslinie bedeutet grundsätzlich, dass eine Instanz während der gesamten Ausführung des Ablaufs existiert und zur Verfügung steht. Oberhalb der Linie steht der Name einer Klasse bzw. das Objektsymbol in der Form: Instanzname:Klassenname. Die Nachrichten werden durch waagerechte Pfeile zwischen den Objektlebenslinien beschrieben. Es gibt folgende Nachrichtenarten:

- Asynchrone Nachrichten werden durch offene Pfeilspitzen gekennzeichnet.
- Synchrone Kommunikation wird durch ausgefüllte Pfeilspitzen gekennzeichnet. Antworten sind optional und werden durch eine gestrichelte Linie mit offener Pfeilspitze dargestellt.
- Geht eine Nachricht verloren bzw. ist der Sender einer Nachricht nicht bekannt (>>gefunden<< Nachricht), so dient ein kleiner ausgefüllter Kreis als End- bzw. Anfangspunkt des Nachrichtenpfeils.

Auf den Pfeilen werden die Nachrichtennamen notiert. Nach der Spezifikation von UML 2.0 [6] wird die Syntax für Nachrichten wie folgende definiert:

- *messageident ::= [attribute=]signal-or-operation-name[(arguments)]
[:return-value] | '*'*
- *arguments ::= argument [, arguments]*
- *argument ::= [parameter-name=]argument-value[attribute=out-parameter-name
[:argument-value]] -*

Die Überlagerung der gestrichelten Lebenslinien durch breite, nicht ausgefüllte (oder graue) senkrechte Balken symbolisiert den Steuerungsfokus. Der Steuerungsfokus ist optional und gibt an, welches Objekt gerade die Programmkontrolle besitzt, d.h. welches Objekt gerade aktiv ist. Lokale Attribute, die beispielsweise als Schleifenzähler o.Ä. verwendet werden können, werden oben links im Diagramm deklariert. Das Erzeugen (Objektkonstruktion) und Entfernen (Objektdestruktion) von Objekten kann in Sequenzdiagrammen ebenfalls dargestellt werden. Die Konstruktion eines neuen Objektes wird durch eine Nachricht, die auf ein Objektsymbol trifft, angezeigt, die Destruktion eines Objektes durch ein Kreuz am Ende der Lebenslinie. Ein Sequenzdiagramm wird mit dem Schlüsselwort **sd** gekennzeichnet.

Für den durch das Diagramm dargestellten zeitlichen Ablauf gelten die folgenden Regeln:

- Die Abfolge von Sende- und Empfangsaktionen auf der gleichen Lebenslinie beschreibt deren zeitliche Abfolge. Hierbei ist aber nur die Anordnung der Ereignisse entscheidend, dagegen lassen sich aus der Länge des vertikalen Abstands zwischen Ereignissen keine Informationen über die abgelaufene Zeit ableiten.
- Die Sendeaktion einer Nachricht findet stets vor der Empfangsaktion für diese Nachricht statt. Auch hier gibt das Diagramm nur die Reihenfolgeninformation wieder, eine Ableitung der Zeitdauer zwischen Senden und Empfangen ist nicht möglich.

Für viele Anwendungen sind die reinen Reihenfolgenangaben von Ereignissen aber nicht ausreichend, so spielen Antwort-, Übertragungs- und Verarbeitungszeiten in Echtzeitsystemen eine entscheidende Rolle. Aus diesem Grund können Lebenslinien und Nachrichten in UML 2.0 mit zusätzlichen Zeitangaben verbunden werden. Es gibt zwei Formen für Zeitangaben: Zeitdauerinformationen und Zeitpunktinformationen.

Ein erstes einfaches Sequenzdiagramm ist in Abbildung 2 gegeben. In diesem Diagramm ist die externe Sicht auf die Abläufe in einer Organisation dargestellt. Wir erkennen hier vier beteiligte Instanzen: ein Objekt von Klasse Organisator, ein Objekt von Klasse Teambesprechung, sowie zwei Instanzen m1 und m2 der Klasse Teammitglied. Das Diagramm visualisiert einen positiven, d.h. gewünschten Ablauf: ein Organisator erzeugt ein Objekt der Klasse Teambesprechung. Das Objekt sendet

zuerst eine Nachricht (Termin bestätigen) an das Objekt m1. m1 schickt dann eine Nachricht OK an das Objekt von Typ Teambesprechung zurück. So funktioniert es auch bei m2. Danach wird eine Nachricht (bestätigt) von Teambesprechung an den Organisator gesendet und das Objekt von Typ Teambesprechung wird gelöscht.

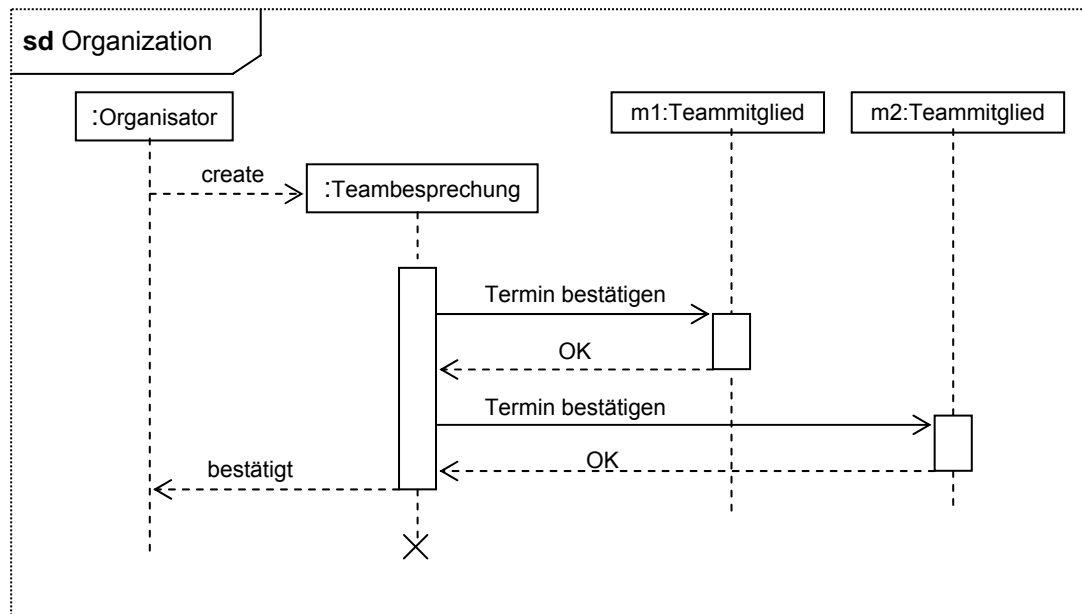


Abbildung 2. Beispiel für ein Sequenzdiagramm

3.2 Strukturierte Sequenzdiagramme

Die lineare Form der Darstellung in Sequenzdiagrammen kann bei komplexen Abläufen schnell zu unübersichtlichen Diagrammen führen. Zusätzlich tauchen auch bestimmte Teilabläufe wie z.B. Initialisierungsphasen in allen Sequenzdiagrammen auf oder aber Abläufe sind bis auf eine kleine Menge von Abweichungen identisch. Aus diesen Gründen bietet UML zwei Mechanismen an, um Sequenzdiagramme besser zu strukturieren und Teilabläufe wieder zu verwenden.

Innerhalb eines Sequenzdiagramms kann durch eine Referenz auf ein anderes Sequenzdiagramm verwiesen werden, das einen Teilablauf beschreibt. Alle Lebenslinien, die von der Referenz überdeckt sind, müssen auch in dem zusätzlichen Diagramm auftreten. Dieses kann aber darüber hinaus noch weitere Lebenslinien enthalten. Durch Referenzen kann eine Interaktionsbeschreibung in beliebig vielen anderen Sequenzdiagrammen wieder verwendet werden.

Operatoren erlauben es auf der anderen Seite, ähnliche Abläufe innerhalb eines Sequenzdiagramms darzustellen. Dabei werden durch die Operatoren diejenigen Abschnitte gekapselt, die die Einzelabläufe voneinander unterscheiden.

Ein Operator wird durch ein Rechteck dargestellt, das horizontal alle Lebenslinien überdeckt, die von diesem Operator betroffen sind und in seiner linken oberen Ecke den Namen des Operators trägt. Alle Kommunikationsaktionen (Senden und Empfangen von Nachrichten), die unter Kontrolle des Operators ablaufen, werden innerhalb des Operator-Rechtecks dargestellt. Der Operator kann wiederum mehrere einzelne Sektionen enthalten, die durch eine gestrichelte waagerechte Linie voneinander abgetrennt sind. Die Interpretation der Einzelsektionen ist unterschiedlich in Abhängigkeit von der Form des Operators. In UML 2.0 gibt es folgende Operatoren.

- opt: Die einfachste Form eines Operators ist der opt-Operator. Mit ihm werden optionale Teilabläufe markiert. Ein opt-Operator umfasst nur eine Sektion.
- seq: Der Seq-Operator definiert eine bestimmte Art der Sequentialisierung der Teilabläufe. (schwache Sequenzialisierung).
- par: Der par-Operator fügt Teilabläufe parallel zusammen. Er erlaubt eine beliebige Vermischung der Sende- und Empfangsaktionen aus den verschiedenen Sektionen. Gefordert wird lediglich, dass die Reihenfolge der Aktionen bezogen auf die Einzelsektionen eingehalten wird.
- strict: Der strict-Operator definiert eine Sequenzialisierung der Teilabläufe die stärker eingeschränkt ist als beim seq-Operator. Die durch den strict-Operator definierte starke Sequenzialisierung verlangt, dass die Abläufe der einzelnen Sektionen vollständig abgeschlossen sind, bevor die Abläufe in den nachfolgenden Sektionen fortgesetzt werden. Somit ist innerhalb des strict-Operators die vertikale Position von Sende- und Empfangsereignissen nicht nur separat für jede einzelne Lebenslinie, sondern für alle Lebenslinien gleichzeitig signifikant.
- loop: Durch den loop-Operator wird die wiederholte Sequenzialisierung der gleichen Abläufe definiert: Der durch den Operator gekapselte Teilablauf kann gar nicht, einmal oder mehrfach nacheinander ausgeführt werden. Der loop-Operator besitzt zwei optionale Argumente, die die minimale und maximale Anzahl an Wiederholungen angeben. Zunächst wird die vorgeschriebene Mindestanzahl an Wiederholungen ausgeführt. Vor jeder weiteren Wiederholung wird dann geprüft, ob die Bedingung wahr ist und ob die maximale Zahl an Wiederholungen noch nicht erreicht wurde, und gegebenenfalls der Operator verlassen.

- alt: Durch einen alt-Operator werden verschiedene alternative Abläufe in einem Sequenzdiagramm zusammengefasst. Ein alt-Operator besitzt zwei oder mehrere Sektionen, die den einzelnen Alternativen entsprechen. Die Auswahl der zu nutzenden Alternative erfolgt auf der Grundlage einer entsprechenden Bedingung, die zur Ausführung der Sektion wahr sein muss. Eine Sektion kann dabei durch else gekennzeichnet werden. Sie wird immer dann ausgewählt, wenn keine der mit expliziten Bedingungen versehenen Sektionen genutzt werden kann.
- ignore: Alle in einem ignore-Operator als Parameter übergebenen Nachrichtentypen werden bei der Ausführung der Interaktion nicht berücksichtigt. Das bedeutet, dass sie den Ablauf nicht beeinflussen und, obwohl nicht explizit angegeben, jederzeit während des Ablaufs auftreten können.
- assert: Durch den assert-Operator kann eine zwingend notwendige Ablauffolge angegeben werden.
- neg: Mit einem Sequenzdiagramm können auch unerwünschte bzw. unzulässige Abläufe spezifiziert werden. Mit dem neg-Operator gekapselte Abläufe entsprechen genau diesen unzulässigen Abläufen.

Es gibt zusätzlich noch ein paar Operatoren, z.B. break, critical, consider, die jedoch in diese Arbeit nicht betrachtet werden.

Die abstrakte Syntax der Interaktion ist durch die Grammatik in Tabelle 1 gegeben.

$$\begin{aligned}
 \textit{Interaction} & ::= \textit{Basic} \\
 & \quad | \textit{CombinedFragment} \\
 \textit{CombinedFragment} & ::= \textit{strict}(\textit{Interaction}, \textit{Interaction}) \\
 & \quad | \textit{seq}(\textit{Interaction}, \textit{Interaction}) \\
 & \quad | \textit{par}(\textit{Interaction}, \textit{Interaction}) \\
 & \quad | \textit{loop}(\textit{Nat}, (\textit{Nat} \mid \infty), \textit{Interaction}) \\
 & \quad | \textit{ignore}(\textit{Messages}, \textit{Interaction}) \\
 & \quad | \textit{alt}(\textit{Interaction}, \textit{Interaction}) \\
 & \quad | \textit{neg}(\textit{Interaction}) \\
 & \quad | \textit{assert}(\textit{Interaction})
 \end{aligned}$$

Tabelle 1. Abstrakte Syntax der Interaktionen (Fragment)

Hier ist ein einfaches Beispiel für den Operator *ref*.

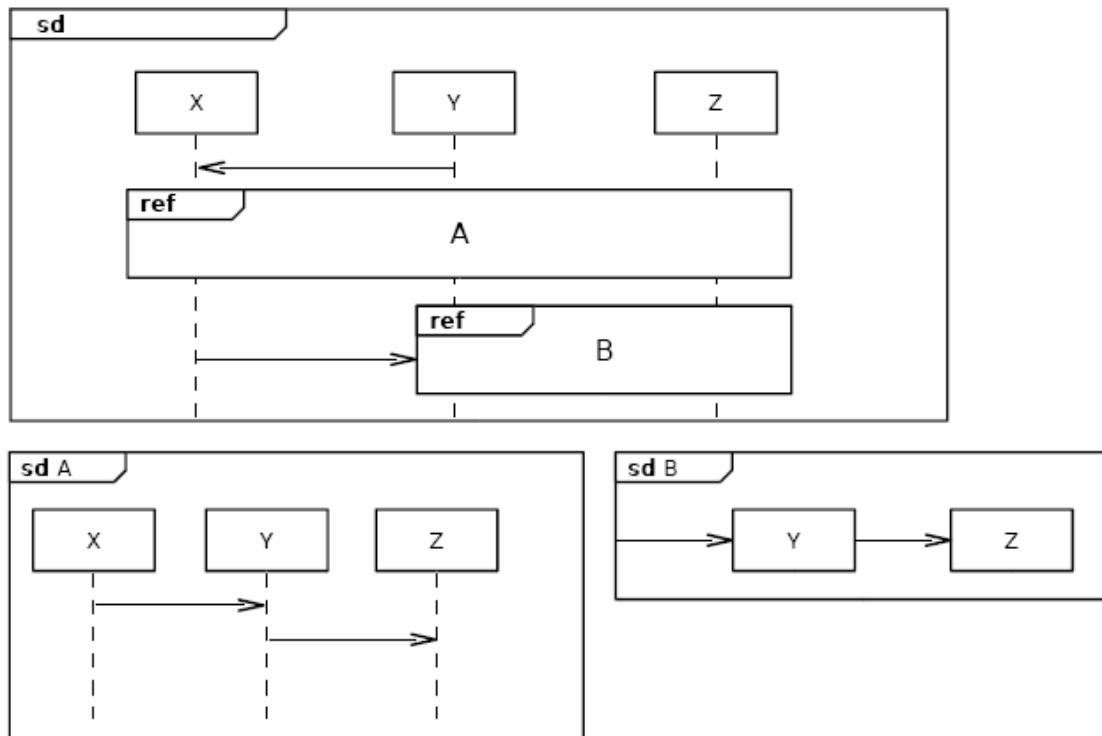


Abbildung 3. Sequenzdiagramm mit Operator *ref*

Hier ist ein ganz einfaches Beispiel für Diagramm mit Referenz. Das Diagramm hat drei Instanzen, x, y, z und zwei Referenzen auf andere Sequenzdiagramme. Wir können die drei Sequenzdiagramme zusammen in ein einziges Sequenzdiagramm zeichnen. Das oben gezeigte Interaktionsdiagramm ist äquivalent zu dem folgenden traditionellen Interaktionsdiagramm:

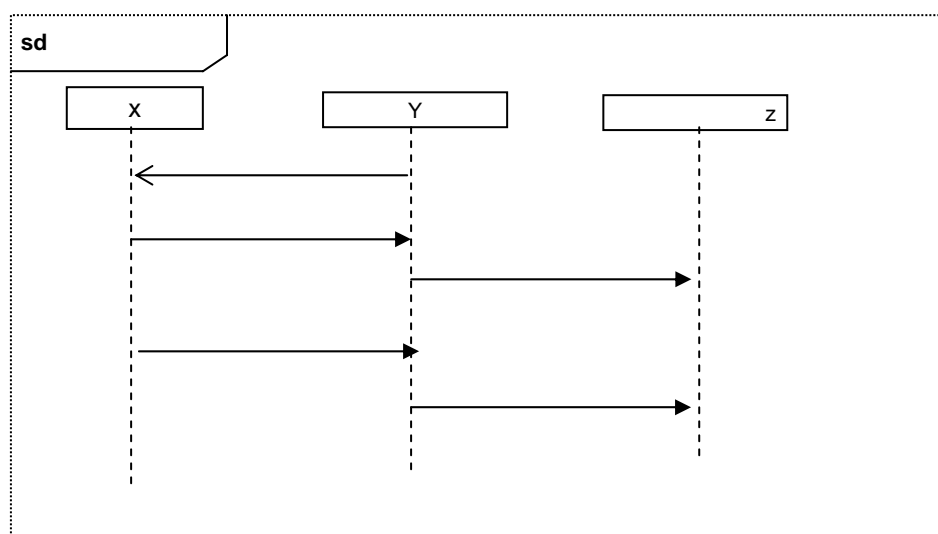


Abbildung 4. Basis-Sequenzdiagramm von Abb. 3

Operatoren können miteinander kombiniert werden. In diesem Fall werden alle anzuwendenden Operatornamen gleichzeitig in dem Operatorrahmen angegeben (z.B. *assert consider* oder *opt alt*).

4. Semantik

In den vorstehenden Abschnitten werden folgende Themen diskutiert: Was ein Sequenzdiagramm ist, welche Komponenten Sequenzdiagramme haben und wie die Komponenten in Diagrammen notiert werden. Wir gehen jetzt noch einen Ebene tiefer und untersuchen, wie die Interaktionen formal beschrieben werden sollen.

4.1 Die Präliminarien für die Semantik der Interaktion

In Abbildung 5 wird ein ganz einfaches Sequenzdiagramm gezeigt.

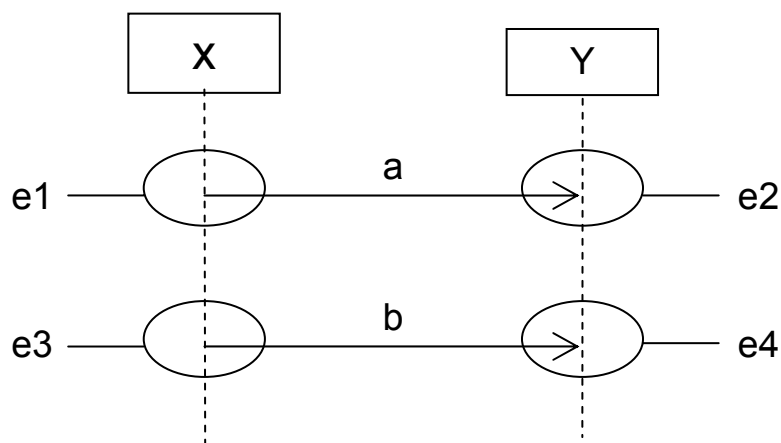


Abbildung 5. Beispiel für einfaches Sequenzdiagramm

In diesem Sequenzdiagramm sind die folgenden Abläufe und nur sie möglich:

$e1 \rightarrow e2 \rightarrow e3 \rightarrow e4$ oder $e1 \rightarrow e3 \rightarrow e2 \rightarrow e4$

Um alle möglichen Abläufe mathematisch formal zu beschreiben, verwendet man partiell geordnete, bezeichnete Multimenge (*pomset*).

Eine partiell geordnete, bezeichnete Multimenge (oder *pomset*) ist eine Klasse $[(X, \leq_x, \lambda_x)]$, wobei X eine vorgegebene Menge von Ereignissen, \leq_x eine partielle Ordnungsrelation und λ_x eine Bezeichnerfunktion sind. Eine Spur ist eine *pomset*, die total geordnet ist. Wir schreiben $lin(p)$ für alle möglichen Linearisierungen einer

$\text{pomset } p. [(X', \leq x', \lambda x')]$ ist ein Element von $\text{lin}([(X, \leq x, \lambda x)])$, wenn und nur wenn, $X'=X, \lambda x' = \lambda x$ and $\leq x \subseteq \leq x'$, wobei $x_1 \leq x' x_2$ oder $x_2 \leq x' x_1$ für alle $x_1, x_2 \in X'$.

Das leere pomset, das durch (ϕ, ϕ, ϕ) repräsentiert wird, ist durch ε gekennzeichnet.

Sei $p = [(X, \leq x, \lambda x)]$ und $q = [(Y, \leq y, \lambda y)]$ und $X \cap Y = \phi$. Die Nebenläufigkeit von p und q , die als $p||q$ geschrieben wird, ist durch die Klasse $[(X \cup Y, \leq x \cup \leq y, \lambda x \cup \lambda y)]$ gegeben. Die Konkatenation von p und q , die als $p;q$ geschrieben wird, ist durch die Klasse $[(X \cup Y, (\leq x \cup \leq y \cup (X \times Y))^*, \lambda x \cup \lambda y)]$ gegeben. Sei \bowtie eine symmetrische Relation der Beschriftungen von p und q . Die \bowtie -Konkatenation von p und q , ist durch $p; \bowtie q$ gegeben, das eine Klasse $[(X \cup Y, (\leq x \cup \leq y \cup \{(x,y) \in X \times Y \mid \lambda_x(x) \bowtie \lambda_y(y)\})^*, \lambda x \cup \lambda y)]$ definiert. Die Konkatenation und \bowtie -Konkatenation sind assoziativ, die Nebenläufigkeit ist assoziativ und kommutativ.

Das pomset, das das Sequenzdiagramm in Abbildung 5 beschreibt, kann wie folgend definiert werden.

$$\begin{aligned}
 & [(\{e_1, e_2, e_3, e_4\}, \{(e_1, e_1), (e_1, e_2), (e_1, e_3)(e_2, e_2), (e_2, e_4)(e_3, e_3), (e_3, e_4)(e_4, e_4)\}, \\
 & \{e_1 \rightarrow \text{snd}(x, y, a), e_2 \rightarrow \text{rcv}(x, y, a), e_3 \rightarrow \text{snd}(x, y, b), e_4 \rightarrow \text{rcv}(x, y, b)\})]
 \end{aligned}$$

Wir definieren zusätzlich noch zwei Domänen für Instanzen I und Nachrichten M . Ein Ereignis e ist entweder in der Form $\text{snd}(s, r, m)$ oder in der Form $\text{rcv}(s, r, m)$. Die beiden Formen repräsentieren das Absenden und den Empfang der Nachricht m mit Sender s und Empfänger r . Hier sind s und r Instanzen. Die Menge von Ereignissen wird durch E bezeichnet. Außerdem definieren wir eine binäre, symmetrische Konfliktrelation \bowtie auf Ereignisse: Wenn eine Instanz aktiv für beide Ereignisse e und e' ist, wird diese Beziehung durch $e \bowtie e'$ bezeichnet. Bei der Repräsentation eines endliches pomsets schreiben wir eine konkretere Notation $\{\text{snd}(s, r, m) \leq \text{rcv}(s, r, m)\}$ statt die $[(\{e_1, e_2\}, \{(e_1, e_1), (e_1, e_2), (e_2, e_2)\}, \{e_1 \mapsto \text{snd}(s, r, m), e_2 \mapsto \text{rcv}(s, r, m)\})]$. In gleicher Weise benutzen wir für die Repräsentation endlicher Spuren des obigen Beispiels die Notation $\text{snd}(s, r, m) \cdot \text{rcv}(s, r, m)$. Die Domäne P beinhaltet alle pomsets $[(E, \leq_E, \lambda_E)]$ mit allen Ereignisse in E . Wir definieren hier eine Funktion: $\text{filter}(M): P \rightarrow \phi P$. Diese Funktion nimmt ein paar Elemente von der Domäne P weg, die eine Nachricht in M haben.

4.2 Die Semantik des positiven Fragments

Wir nennen die Interaktionen ohne Auftreten der Negation und Assertion das positive Fragment. Die positive Befriedigungsrelation (Eng. Positiv Satisfactions) zwischen Spuren und Interaktionen ist durch $t \models_p S$ gekennzeichnet, wobei t eine Spur und S

eine Interaktion des positiven Fragments ist. Die Semantik des positiven Fragments ist induktiv in Tabelle 2 definiert; dabei steht B für die Menge der (nicht durch Operatoren zusammengesetzten) Basis-Interaktionen, S , S_1 und S_2 sind Interaktionen, M steht für Nachrichten, m und n sind natürliche Zahlen.

$$\begin{aligned}
 t \models_p B & \text{ if } t \in \text{lin}(B) \\
 t \models_p \text{strict}(S_1, S_2) & \text{ if } \exists t_1, t_2 . t = t_1 ; t_2 \wedge t_1 \models_p S_1 \wedge t_2 \models_p S_2 \\
 t \models_p \text{seq}(S_1, S_2) & \text{ if } \exists t_1, t_2 . t \in \text{lin}(t_1 ; \bowtie t_2) \wedge t_1 \models_p S_1 \wedge t_2 \models_p S_2 \\
 t \models_p \text{par}(S_1, S_2) & \text{ if } \exists t_1, t_2 . t \in \text{lin}(t_1 \parallel t_2) \wedge t_1 \models_p S_1 \wedge t_2 \models_p S_2 \\
 t \models_p \text{loop}(0, 0, S) & \text{ if } t = \varepsilon \\
 t \models_p \text{loop}(0, n + 1, S) & \text{ if } t = \varepsilon \vee t \models_p \text{seq}(S, \text{loop}(0, n, S)) \\
 t \models_p \text{loop}(m + 1, n + 1, S) & \text{ if } t \models_p \text{seq}(S, \text{loop}(m, n, S)) \\
 t \models_p \text{loop}(m, \infty, S) & \text{ if } \exists n \geq m . t \models_p \text{loop}(m, n, S) \\
 t \models_p \text{ignore}(M, S) & \text{ if } \exists t_1 . t_1 \in \text{filter}(M)(t) \wedge t_1 \models_p S \\
 t \models_p \text{alt}(S_1, S_2) & \text{ if } t \models_p S_1 \vee t \models_p S_2
 \end{aligned}$$

Tabelle 2. Semantik des positiven Fragments

4.3 Die Negationen

In dieser Arbeit werden wir uns auf die Negation konzentrieren. In der UML 2.0 Spezifikation sind manche Features der Interaktionssprache nicht eindeutig spezifiziert, z.B. die Negation. In [6] wird der Operator `neg` folgendermaßen spezifiziert:

The interaction Operator `neg` designates that the Combined Fragment represents traces that are defined to be invalid. The set of traces that defined a negative Combined Fragment is equal to the set of traces given by its (sole) operand, only that this set is a set of invalid rather than valid traces. All Interaction Fragments that are different from Negative are considered positive meaning that they describe traces that are valid and should be possible.

UML 2.0 Interaktionen beschreiben positive und negative Spuren des Auftretens von Ereignissen. Die Benutzung der monadischen Operatoren `neg(-)` und `assert(-)` führt zu negativen Spuren. Die positiven und negativen Spuren, die durch eine Interaktion definiert sind, decken nicht alle mögliche Interaktionen ab. Die übrigen Spuren werden ergebnislos (Eng. inconclusive) genannt. Um die Ungenauigkeiten der Spezifikation zu vermeiden, versucht man eine klarere Spezifikation für Operator `neg`

zu definieren. Störrle bedenkt drei verschiedene Interpretationen für Operator $\text{neg}(s)$. (1) „not the [valid] traces of s “. (2) „Anything but the [valid] traces of s “. (3) The negative traces of S to be the positive traces for $\text{neg}(s)$ [7]. Haugen und Stølen interpretieren hingegen, dass die positive Spuren von $\text{neg}(S)$ nur aus der leeren Spur bestehen [3]. In dieser Arbeit werden wir die Definition von Haugen und Stølen benutzen.

4.4 Die Semantik des negativen Fragments

Die Semantik einer verneinte Interaktion $\text{neg}(S)$ ist wie folgend definiert. Die positiven Spuren von $\text{neg}(S)$ sind solche Spuren, die nicht positiv für S sind und die negativen Spuren sind die Spuren, die positiv für S sind. Derartige Definition schließen jedoch die ergebnislosen (inconclusive) Spuren aus. Im Allgemeinen müssen wir die positiven, negativen und inconclusive Abläufe unterscheiden. Wir schreiben \models_n , wenn t negativ S befriedigt. Die induktive Definition für die Relation \models_n ist in Tabelle 3 gezeigt, wobei B für die Menge der Basis-Interaktionen steht, S , S_1 und S_2 sind Interaktionen, M steht für Nachrichten, m und n sind natürliche Zahlen.

Wir definieren \models_n besonders für alle kombinierten Interaktionsfragmente. Sie entsprechen der Definition von Haugen und Stølen. Wir betrachten die leere Spur als positiv für $\text{neg}(S)$. Für die kombinierte Fragments $\text{strict}(-, -)$ und $\text{seq}(-, -)$ sind nur solche Spuren negativ, die negativ für den erste Operand sind oder positive für den erste Operand aber negativ für den zweite Operand sind.

Ein sehr interessanter Fall ist gegeben durch folgendes Beispiel. Sei B_i die Basis-Interaktion $\{\text{snd}(s_i, r_i, m_i) \leq \text{rcv}(s_i, r_i, m_i)\}$ ($i = 1, 2, 3$), wobei alle m_i verschieden sind und t_i die Spuren $\text{snd}(s_i, r_i, m_i) \cdot \text{rcv}(s_i, r_i, m_i)$ ($i = 1, 2, 3$) bedeuten.

- $t_2 \models_p \text{strict}(\text{neg}(B_2), B_2)$
- $t_2 \models_n \text{strict}(\text{neg}(B_2), B_2)$

Von den obenstehenden zwei Formeln können wir ableiten, dass t_2 gleichzeitig positiv und negativ für die Interaktion $\text{strict}(\text{neg}(B_2), B_2)$ ist. Wir nennen $\text{strict}(\text{neg}(B_2), B_2)$ deshalb eine überspezifizierte Interaktion.

$$\begin{aligned}
 t \models_p \text{neg}(S) & \text{ if } t = \varepsilon \\
 t \models_p \text{assert}(S) & \text{ if } t \models_p S \\
 t \models_n \text{strict}(S_1, S_2) & \text{ if } \exists t_1, t_2 . t = t_1 ; t_2 \wedge (t_1 \models_n S_1 \vee (t_1 \models_p S_1 \wedge t_2 \models_n S_2)) \\
 t \models_n \text{seq}(S_1, S_2) & \text{ if } \exists t_1, t_2 . t \in \text{lin}(t_1 ;_{\infty} t_2) \wedge (t_1 \models_n S_1 \vee (t_1 \models_p S_1 \wedge t_2 \models_n S_2)) \\
 t \models_n \text{par}(S_1, S_2) & \text{ if } \exists t_1, t_2 . t \in \text{lin}(t_1 \parallel t_2) \wedge ((t_1 \models_n S_1 \wedge t_2 \models_n S_2) \vee \\
 & (t_1 \models_n S_1 \wedge t_2 \models_p S_2) \vee (t_1 \models_p S_1 \wedge t_2 \models_n S_2)) \\
 t \models_n \text{loop}(0, n + 1, S) & \text{ if } t \models_n \text{seq}(S, \text{loop}(0, n, S)) \\
 t \models_n \text{loop}(m + 1, n + 1, S) & \text{ if } t \models_n \text{seq}(S, \text{loop}(m, n, S)) \\
 t \models_n \text{loop}(m, \infty, S) & \text{ if } \exists n \geq m . t \models_n \text{loop}(m, n, S) \\
 t \models_n \text{ignore}(M, S) & \text{ if } \exists t_1 . t_1 \in \text{filter}(M)(t) \wedge t_1 \models_n S \\
 t \models_n \text{alt}(S_1, S_2) & \text{ if } t \models_n S_1 \wedge t \models_n S_2 \\
 t \models_n \text{neg}(S) & \text{ if } t \neq \varepsilon \wedge t \not\models_p S \\
 t \models_n \text{assert}(S) & \text{ if } t \not\models_p S
 \end{aligned}$$

Tabelle 3. Erweiterte Semantik für die Negation

5. Verbesserung der Negation

Die nicht klassische Interpretation der Negation ist schwierig zu handhaben. Vor allem werden zwei Befriedigungsrelationen, eine positive und eine negative, gebraucht. Es ist schwer zu entscheiden, ob eine Spur positiv, inconclusive oder negativ für eine Interaktion ist. Die Negation ist natürlich nicht nötig für die positive Befriedigungrelation. Für die Ableitung der negativen Befriedigung wird die Negation natürlich gebraucht. Die kann aber durch eine klassische Version ersetzt werden. Wir führen jetzt die Sprache der Interaktion in klassische Sinne ein.

5.1 σ -Funktionen

Wir definieren hierfür eine Funktion σ , die einer Interaktion genau eine Interaktion des positiven Fragments zuweist. Die Funktion σ ist induktiv wie in Tabelle 5 definiert; dabei steht B wieder für die Menge der Basis-Interaktionen, S , S_1 und S_2 sind Interaktionen, M steht für Nachrichten, m ist eine natürliche Zahl, \bar{n} ist eine natürliche Zahl oder ∞ .

$$\begin{aligned}
\sigma(B) &= B \\
\sigma(\text{strict}(S_1, S_2)) &= \text{strict}(\sigma(S_1), \sigma(S_2)) \\
\sigma(\text{seq}(S_1, S_2)) &= \text{seq}(\sigma(S_1), \sigma(S_2)) \\
\sigma(\text{par}(S_1, S_2)) &= \text{par}(\sigma(S_1), \sigma(S_2)) \\
\sigma(\text{loop}(m, \bar{n}, S)) &= \text{loop}(m, \bar{n}, \sigma(S)) \\
\sigma(\text{ignore}(M, S)) &= \text{ignore}(M, \sigma(S)) \\
\sigma(\text{alt}(S_1, S_2)) &= \text{alt}(\sigma(S_1), \sigma(S_2)) \\
\sigma(\text{neg}(S)) &= \text{skip} \\
\sigma(\text{assert}(S)) &= \sigma(S)
\end{aligned}$$

Tabelle 4. Die Definition für die σ Funktion

Aus Tabelle 2 und Tabelle 4 ist leicht abzuleiten, dass $t \models_p S$ wenn und nur wenn $t \models_p \sigma(S)$. Aber eine binäre Logik ohne Negation ist nicht genug. Allerdings reicht eine binäre Logik mit klassischer Negation aus. Wir fügen hierfür einen Operator $\text{not}(-)$ zum positiven Fragment der Interaktion hinzu. Diese Interaktion ist eine sogenannte Interaktion im klassischen Sinne. Die Operatoren $\text{neg}(-)$ und $\text{assert}(-)$ werden entfernt und der Operator $\text{not}(-)$ wird hinzugefügt. Die positive Semantik der Interaktion im klassischen Sinne ist gegeben durch die Semantik für das positive Fragment von UML 2.0 Interaktion in Tabelle 2 und

$$t \models_p \text{not}(S) \quad \text{if } t \not\models_p S$$

Wir führen weitere Abkürzung hinzu:

Any = $\text{ignore}(M, \text{skip})$

None = $\text{not}(\text{Any})$

and(S_1, S_2) = $\text{not}(\text{all}(\text{not}(S_1), \text{not}(S_2)))$

5.2 ν -Funktionen

Wir definieren eine Funktion ν , die einer Interaktion genau eine Interaktion im klassischen Sinne zuweist. Die Funktion σ ist induktiv wie in Tabelle 6 definiert, wobei B für die Menge der Basis-Interaktion steht, S, S_1 und S_2 sind Interaktionen, M steht für Nachrichten, m ist eine natürliche Zahl, \bar{n} eine natürliche Zahl oder ∞ .

$$\begin{aligned}
 \nu(B) &= \text{None} \\
 \nu(\text{strict}(S_1, S_2)) &= \text{alt}(\text{strict}(\nu(S_1), \text{Any}), \text{strict}(\sigma(S_1), \nu(S_2))) \\
 \nu(\text{seq}(S_1, S_2)) &= \text{alt}(\text{seq}(\nu(S_1), \text{Any}), \text{seq}(\sigma(S_1), \nu(S_2))) \\
 \nu(\text{par}(S_1, S_2)) &= \text{alt}(\text{par}(\nu(S_1), \nu(S_2)), \text{par}(\nu(S_1), \sigma(S_2)), \text{par}(\sigma(S_1), \nu(S_2))) \\
 \nu(\text{loop}(m, \bar{n}, S)) &= \text{and}(\text{loop}(m, \bar{n}, \nu(S)), \text{not}(\text{skip})) \\
 \nu(\text{ignore}(M, S)) &= \text{ignore}(M, \nu(S)) \\
 \nu(\text{alt}(S_1, S_2)) &= \text{and}(\nu(S_1), \nu(S_2)) \\
 \nu(\text{neg}(S)) &= \text{and}(\sigma(S), \text{not}(\text{skip})) \\
 \nu(\text{assert}(S)) &= \text{not}(\sigma(S))
 \end{aligned}$$

 Tabelle 5. Die Definition für die ν Funktion

Sei S eine UML 2.0 Interaktion und t eine Spur. Dann ist $t \models_n S$, dann und nur dann, wenn, $t \models_p \nu(S)$. Um dies zu beweisen, führen wir eine partielle Ordnung \leq in UML 2.0 Interaktionen ein, wobei S, S_1 und S_2 beliebige Interaktionen, und m und n natürliche Zahlen sind.

$$\begin{array}{ll}
 \text{skip} \leq S & S \leq \text{neg}(S) \\
 S_1 \leq \text{strict}(S_1, S_2) & S_2 \leq \text{strict}(S_1, S_2) \\
 S_1 \leq \text{seq}(S_1, S_2) & S_2 \leq \text{seq}(S_1, S_2) \\
 S_1 \leq \text{par}(S_1, S_2) & S_2 \leq \text{par}(S_1, S_2) \\
 S_1 \leq \text{alt}(S_1, S_2) & S_2 \leq \text{alt}(S_1, S_2) \\
 S \leq \text{ignore}(M, S) & S \leq \text{assert}(S) \\
 \text{seq}(S, \text{loop}(m, n, S)) \leq \text{loop}(m, n+1, S) & \text{loop}(m, n, S) \leq \text{loop}(m, \infty, S)
 \end{array}$$

 Tabelle 6. Partielle Ordnung \leq an UML 2.0 Interaktionen

Zusammengefasst haben wir zwei Resultate:

$$t \models_p S \text{ falls } t \models_p \sigma(S)$$

$$t \models_n S \text{ falls } t \models_p \nu(S)$$

Dabei ist S eine beliebige UML 2.0 Interaktion, $\sigma(S)$ eine Interaktion des positiven Fragments und $\nu(S)$ eine Interaktion im klassischen Sinn. Mittels der zwei Transformationen σ und ν können wir ein Testen auf negative Befriedigung vermeiden.

6. Implementierung und Verfeinerung

Wir haben eine formale Semantik für Interaktionen und führen jetzt die Notation für die Implementierung einer Interaktion durch einen Prozess ein.

Ein Prozess I ist eine Implementierung einer Interaktion S , geschrieben $I \models S$, falls

1. Es existiert ein $t \in \text{lin}(I)$ mit $t \models_p S$, und
2. $t \not\models_n S$ für alle $t \in \text{lin}(I)$.

Eine Interaktion ist implementierbar, wenn es einen Prozess I mit $I \models S$ gibt. Für jede Interaktion S existiert eine Spur t mit $t \models_p S$. Zwei Interaktionen S_1 und S_2 sind äquivalent, gekennzeichnet durch $S_1 \equiv S_2$, wenn für alle Prozesse I gilt, $I \models S_1$ wenn und nur wenn $I \models S_2$.

Eine Interaktion S' verfeinert eine Interaktion S , geschrieben $S \rightsquigarrow S'$, wenn alle Implementierung für S' auch eine Implementierung für S sind. Das heißt, $I \models S'$ impliziert $I \models S$ für alle Implementierung I .

Ein Beispiel für eine Interaktionsverfeinerung ist wie folgend gezeigt.

$\text{alt}(S_1, S_2) \rightsquigarrow S_i$, für $i = 1, 2$

7. Zusammenfassung

Im Dokument UML 2.0 Spezifikation sind einige der Features der Interaktionssprache nicht eindeutig spezifiziert, z.B. die Negation. Das Ziel dieser Arbeit ist der Versuch, die Lösung des Problems zu beschreiben. Deshalb haben wir zuerst die Syntax und Semantik der Interaktion formal beschrieben. Besonders haben wir die positiven, negativen und inconclusiven Abläufe für Interaktionen klassifiziert. Schließlich versuchten wir durch die Einführung von zwei Funktionen, σ und ν eine klassische Interpretation des Negationsbegriffs einzuführen, die das Testen auf negative Befriedigung überflüssig macht.

8. Literaturverzeichnis

1. Maria Victoria Cengarle und Alexander Knapp. UML 2.0 Interactions: Semantics and Refinement. In Jan Jürgens et. Al., editors, *Critical Systems Development with UML (CSDUML'04, Proceedings)*. To appear, 2004
2. Tom Pender. *UML Bible*. Wiley Publishing, 2004
3. Øystein Haugen and Ketil Stølen. STAIRS—Steps to Analyze Interactions with Refinement Semantics. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *Proc. 6th Int. Conf. Unified Modeling Language (UML'03)*, volume 2863 of *Lect. Notes Comp. Sci.*, pages 388–402. Springer, Berlin, 2003.
4. Marc Born, Eckhardt Holz und Olaf Kath. *Softwareentwicklung mit UML 2*. ADDISON-WESLEY, 2004
5. Bernd Oestereich. *Objektorientierte Softwareentwicklung, Analyse und Design mit der UML 2.0*. Oldenbourg, 2004
6. Object Management Group. *Unified Modeling Language Specification, Version 2.0 (Superstructure)*. Adopted draft, OMG, 2003. <http://www.omg.org/cgi-bin/doc?ptc/03-08-02>.
7. Harald Störrle. Assert, Negate and Refinement in UML-2 Interactions. In Jan Jürgens, Bernhard Rumpe, Robert France, and Eduardo B. Fernandez, editors, *Proc. Wsh. Critical Systems Development with UML (CSDUML'03)*, San Francisco, 2003. Technische Universität München, Technical report TUM-I0317.
8. <http://www3.informatik.uni-erlangen.de/Lehre/UML-Seminar/SS2003/vortrag3.pdf>
9. <http://www4.in.tum.de/~prehofer/vorlesung-04/modellierung2.pdf>
10. <http://tfs.cs.tu-berlin.de/lehre/SS04/GRA-IOSIP/Folien/bjoern.pdf>
11. <http://www-inf.fh-reutlingen.de/dbweb/content/DM/UML.pdf>
12. <http://www.pst.informatik.uni-muenchen.de/personen/stoerrle/V/JVLC.pdf>
13. http://de.wikipedia.org/wiki/Unified_Modeling_Language
14. http://www.big.tuwien.ac.at/teaching/offer/ws04/ooae_vu/uml_work1-54.pdf