BERUFSAKADEMIE | MANNHEIM
university of cooperative education | staatliche studienakademie

Vortrag

Die Unified Modeling Language 2
„Next Generation ?“

Seminar Informationstechnik
Prof. Tischhauser
Gehalten von Andreas Leder

Gliederung

1 Einführung

1.1 Geschichtliches zur UML

In den 90er Jahren gab es eine Vielzahl an objektorientierten Methoden und Modellierungsmöglichkeiten über Raumbaughs OMT, Wirfs-Brocks RDD, Cherrys SCOOP oder Jacobsens OOSE. Jeder potenzielle Anwender stand vor dem Problem welche Notation wohl die geeignete ist und vor allem welche auch noch in 2 Jahren am Markt ist. Durch Vereinigung einiger Ansätze und verschwinden anderer etablierte sich 1996 eine, die UML in der Version 0.9. Sie entstand aus der Unified Method von Rumbaugh\Booch und dem Ansatz OOSE nach Jacobsen, auch genannt die 3 Amigos. Mit der Version 1.2 sollte die Übergabe der Urheberrechte an die Objekt Management Group stattfinden was allerdings solang gedauert hat dass schon eine neue Version 1.3 im Jahre 1999 zur Verfügung stand. Ab hier liegt das Copyright auch bei der OMG. Die im Jahre 2001 veröffentlichte Version 1.4 ist Grundlage für die wohl größte und umfassendste Überarbeitung zur Version 2.0 welche mit 1.5 Jahren Verspätung im Herbst 2004 durch die OMG endgültig verabschiedet werden.

1.2 Was ist die UML…und was will sie nicht sein?

Sie ist erst einmal die heute am weitesten verbreitete Notation um Softwaresystem zu analysieren und zu entwerfen. Sie dient zur Modellierung, Dokumentation, Spezifizierung -und Visualisierung. Sie unterstützt dynamische wie statische Modelle zur Analyse, Design und Architektur vor allem in objektorientierten Vorgehensweisen. Genau in diesen Anforderungen sowie durch den schnellen Fortschritt in der Softwareentwicklung liegt auch begründet dass die UML nie vollständig und in sich stimmig sein wird. Vielfach entstehen auch Missverständnisse durch eine falsche Erwartungshaltung an die UML. Daher sollte man sich immer folgende Punkte vor Augen halten im Bezug auf die UML. Dass die UML

- Nicht perfekt,
- Nicht vollständig,
- Keine Programmiersprache,
- Keine rein formale Sprache,
- Nicht spezialisiert auf ein Anwendungsgebiet,
- Kein vollständiger Ersatz für Textbeschreibung und vor allem
- Keine Methode und auch kein Vorgehensmodell

ist und dies alles auch nicht sein will!

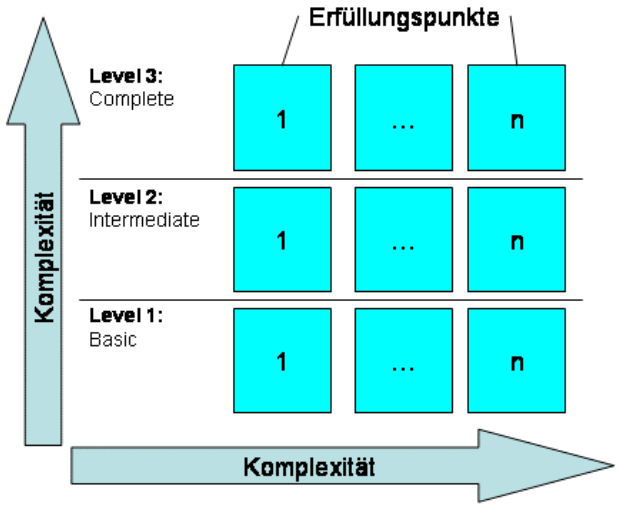
2 Allgemeines in der UML „Next Generation“

Der erste Unterschied welcher bei der neuen UML auffällt ist das sie als 2 bezeichnet wird und nicht mehr als 1.X. Es hat also ein kompletter Versionswechsel statt gefunden bedingt durch eine lange Liste von Wünschen und Anforderungen die durch den Praxiseinsatz der vorhergehenden Versionen entstanden sind. Da vieles davon nicht mehr sinnvoll in die alte UML integrierbar war wurde die UML von Grund auf neu aufgesetzt. Allgemein ist zu sagen das die UML für die Tool-Hersteller und Anwender eine bessere Skalierbarkeit bietet im Hinblick darauf **WIE** und **WIEVIEL** UML in Projekten eingesetzt werden soll. Dies wird erreicht durch die Einführung so genannter Compliance Level.
Die UML ist in der neuen Version kompakter und schlüssiger geworden. Durch anpassen grundlegender Konzepte in der Struktur ist ein wichtiger Schritt in Richtung Widerverwend-

barkeit und Straffung der Konzepte geschafft. Die UML 2 ist definiert durch die Infrastructure und die Superstructure. Hierbei sind die grundlegenden Sprachkonstrukte und die Basisarchitektur in der Infrastructure definiert. Aufbauend auf dieser Architektur legt die Superstructure nun die Diagrammnotationen sowie die Semantik der Sprache fest.

2.1 „Wenig UML ist immer noch UML“

Der Sprachumfang der UML 2 ist in Erfüllungsebenen (Compliance Levels) sowie Punkte (Compliance Points) gegliedert. Die Idee hierzu ist in der Datenbankwelt mit ihrer Sprache SQL zu finden. SQL ist in verschiedene (4) Levels aufgegliedert die den Umfang der zu verwendenden Sprachkonstrukte vorgibt. Die UML selbst ist nun in 3 Schichten unterteilt. Die erste Ebene ist die „basic“ darauf setzen 2 Schichten auf „intermediate“ und als oberste „complete“. Jede dieser Ebenen hat ein eigenes Komplexitätsniveau. Zusätzlich zu den Ebenen sind wie oben schon erwähnt Erfüllungspunkte eingeführt wurden. Jede Ebene ist hierbei zusätzlich noch einmal in Erfüllungspunkte untergliedert. Ein Erfüllungspunkt stellt eine inhaltlich gruppierte Sammlung von Notationskonstrukten dar.



Compliance Level 1

Jeder Toolhersteller muss nun zu den Erfüllungspunkten auch noch angeben wie hoch der Grad der Erfüllung (Compliance Option) ist. Unterstützt das Tool in einem Punkt die Notationselemente nicht oder nicht standardkonform so ist er als „nicht erfüllt“ auszuzeichnen. Werden Notationselemente teilweise oder teilweise standardkonform unterstützt so ist er „teilweise erfüllt“. „Vollständig erfüllt“ heißt dann alle Notationselemente sind standardkonform unterstützt. Erfüllt das Tool nun auch das Standardaustauschformat für die Notationselemente so hat es den Grad „austauschbar“.

Was sind die Ziele hinter all dem Compliance…?

- Keine Überforderung des Anwenders durch vollen Sprachumfang
- Je nach Projekteinsatz ist es nun möglich die benötigten Notationselemente flexibler auszuwählen
- Toolhersteller können UML 2 schrittweise einführen
- Toolhersteller können Tools an bestimmte Zielmärkte anpassen

3.3 Die Diagramme – Ein Überblick

Die neue UML unterstützt 13 Diagrammarten, effektiv sind also 3 Diagrammtypen dazu gekommen im Gegensatz zur alten UML. Nachfolgend noch mal eine kurze Übersicht der unterstützen Diagramme, die rot hervorgehobenen sind die neu eingeführten:

Die Strukturdiagramme

- Klassendiagramm
- Objektdiagramm
- Komponentendiagramm
- Verteilungsdiagramm
- Kompositionsstrukturdiagramm
- Paketdiagramm

Die Verhaltensdiagramme

- Aktivitätsdiagramm
- Use-Case Diagramm
- Zustandsautomat

Die Interaktionsdiagramme

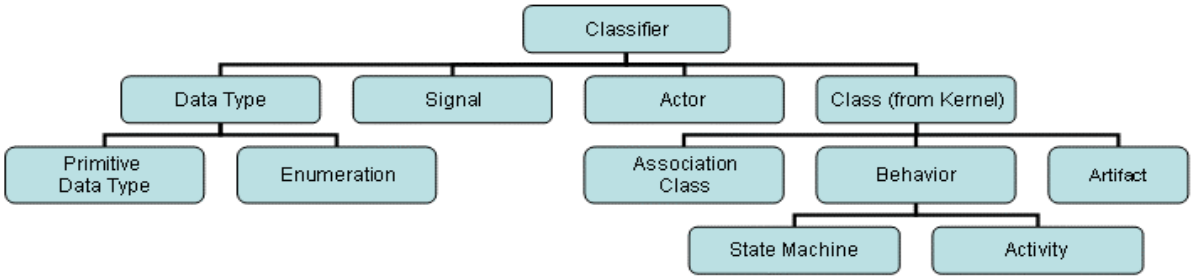
- Sequenzdiagramm
- Interaktionsübersichtsdiagramm
- Kommunikationsdiagramm
- Timingdiagramm

Hierzu muss man sagen dass das Kommunikationsdiagramm nur vom Namen her neu ist denn es ist das alte Kollaborationsdiagramm ohne wesentliche Änderungen. Allgemein kann man sagen das einige Diagrammarten gravierender Änderungen erfahren haben als andere. Manche wurden komplett neu überarbeitet andere wiederum wurden nur um einige Unsauberkeiten bereinigt. Eine erwähnenswerte Neuerung fand bei den Verhaltensdiagrammen statt. Erweiternd oder auch ergänzend ist es nun möglich Verhalten tabellarisch darzustellen was zur Testfall –Generierung oder zur Vollständigkeitsprüfung heran gezogen werden kann. In den nachfolgenden Kapiteln sollen die Diagramme nun einmal in ihren Änderungen näher vorgestellt werden. Hauptaugenmerk liegt hierbei jedoch hauptsächlich auf den Diagrammen mit den größten Änderungen und natürlich auf den 3 wirklich neuen. Dazu wird vorher im Kapitel 2.2.1 aber noch das Konzept des „Classifiers“ vorgestellt dem eine große Rolle in der UML 2 zukommt.

3.4 Classifier

Wie unschwer zu erkennen ähnelt der Begriff Classifier sehr dem Begriff Klasse. Dies kommt nicht von ungefähr denn in der UML 2 wird das Grundkonzept der Klasse aufgegriffen und noch um einen Schritt weiter abstrahiert. Es ist hierbei zu erwähnen dass auch in UML 1.x Classifier vorhanden waren aber in der UML 2 die Konsequenz des Konzeptes absolut umgesetzt wird. Innerhalb des Sprachdesigns der UML spielt er eine zentrale Rolle auf die immer wieder zurückgegriffen wird. Der Classifier muss hier mehr als gedankliches Konstrukt als ein nutzbares, graphisches Modellelement gesehen werden. Eigenschaften von Modellelemen-

ten werden hierbei bereits auf der abstrakten Ebene des Classifiers definiert und stehen so unterschiedlichsten Elmenten einheitlich zur Verfügung. Betrachten wir einmal beispielhaft eine Zuordnungsbeziehung. Solch eine Beziehung ordnet dem Namen nach eindeutig eine Sache einer anderen Sache zu. Auf der Ebene von Modellelementen der UML könnten solche Sachen ja zum Beispiel eine Klasse, Aktivitäten oder auch Operationen sein. Nun ist es möglich eine Klasse anderen Klassen zuzuordnen oder auch eine Aktivität einer Operation zuzuordnen. Also kann jedes dieser Modellelemente eine Zuordnungsbeziehung besitzen wenn der Modellierer das will. Nun macht es keinen Sinn eine Zuordnungsbeziehung für jedes Element extra zu definieren denn das hilft immer noch nicht in dem Moment wenn eine Aktivität einer Operation zugeordnet werden soll. Daher ist einfacher alle Elemente auf der Ebene von Zuordnungen gleich zu betrachten oder anders ausgedrückt jedes dieser 3 Elemente ist ein Classifier mit der Eigenschaft das er anderen zugeordnet werden kann. Auf diese Art und Weise sind nun im gesamten Metamodell alle Modellelemente auf abstrakter Ebene als Classifier definiert die bestimmte Eigenschaften haben. Jedes grafische Modellelement ist also auf abstrakter Ebene im Metamodell in Form eines oder auch mehrer Classifier definiert. Hieraus ergibt sich im Metamodell eine Classifierhierachie in der Classifier auch von anderen Classifier Verhalten oder Eigenschaften erben können und erweitern. Nachfolgend ein Teil der Hierarchie wie sie im Metamodell definiert ist einmal dargestellt.



Der im Bild dargestellte Classifier „Class (from Kernel)“ soll an dieser Stelle noch kurz vorgestellt werden. Dieser Classifier ist die direkte Ausprägung der Klasse wie sie auch schon in der UML 1.X in den Klassendiagrammen angewandt wird. Es gibt noch 2 weitere Ausprägungen „Class (from Communication)“ und „Class (from StructuredClasses)“. „Class (from StructuredClasses)“ zum Beispiel erweitert den Klassenbegriff um die Möglichkeit eine interne Struktur und Ports in einer Klasse zu strukturieren. Allen Classifiern gemein ist das sie Spezialisierbar und in der anderen Richtung generalisierbar sowie strukturierbar sind.

4 Die „Next Generation“ Diagramme

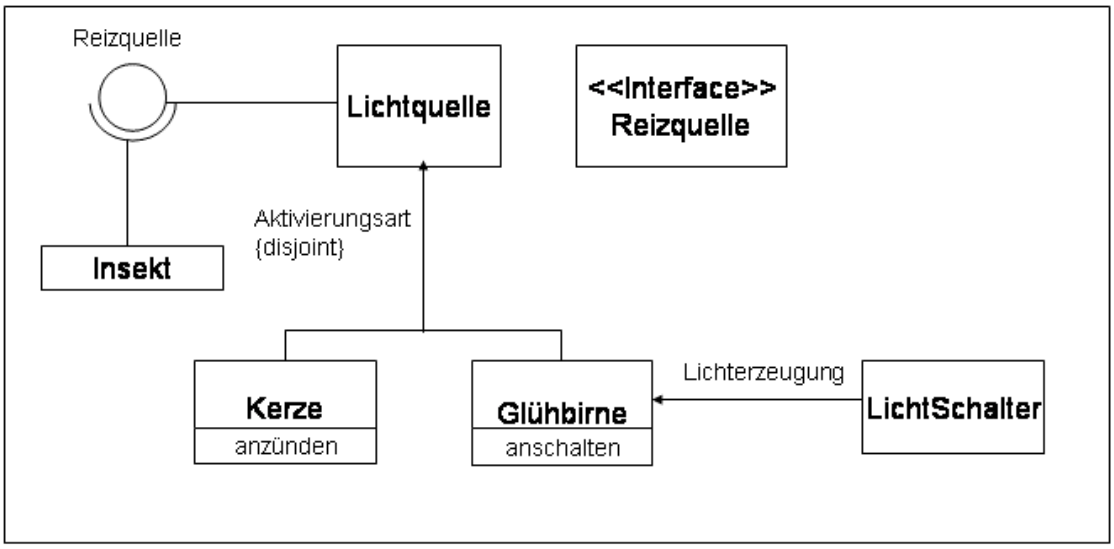
In diesem Kapitel nun sollen die direkten für den Modellierer zukommenden Änderungen, Erweiterungen und somit hoffentlich auch Verbesserungen aufgezeigt werden. Hierbei ist es in 2 große Teile unterteilt frei folgend den Aspekten einer statischen und einer dynamischorientierten Modellierung. Hierbei wird die Tiefe sehr stark variieren, je nachdem wie stark ein Diagramm Veränderungen erfahren hat da dies keine Einführung in UML sein soll sondern die Änderungen in der neuen UML 2 aufzeigen soll. Ausnahme soll hierbei das Klassendiagramm sein welches wenige Änderungen erfahren hat aber mit das wichtigste ist. Ein einführendes Wort zum sinnvollen Einsatz und was ein Diagramm aussagt soll trotz allem genannt sein.

3.1 Strukturdiagramme

Die 6 Strukturdiagramme erlauben es dem geneigten Modellierer sein System aus verschieden statischen Blickwinkeln zu betrachten die von der „einfachen“ Aufbaustrukturmodellierung einzelner Klassen bis hin zur Gliederung vollständiger Architekturen reicht. Das Paketdiagramm wurde unverändert aus den Vorgängerversionen übernommen. Das Objektdiagramm hat auch keine wesentlichen Änderungen erfahren außer der Beseitigung der Stereotype <<copy>> und <<become>>.

3.1.1 Das Klassendiagramm

„Wie sind Daten und –Verhalten in meinem System im Detail strukturiert“



Das Klassendiagramm gibt mir also Überblick über die statische Struktur des zu entwerfenden Systems mit seinen relevanten Strukturzusammenhängen und Datentypen. Es ist normalerweise das unverzichtbarste Modellierungsartefakt. Aus dem Blickwinkel der UML ist es der Kern der gesamten Modellierungssprache denn hier werden schon die wichtigsten grafischen Symbole und ihre Bedeutung ein. Der Wechsel von der UML 1.X zur hat UML 2 für den Modellierer nur wenig Neues außer der Bereinigung einiger Unsauberkeiten. Hier die Änderungen kurz zusammen gefasst einige Änderungen sind im obigen Beispiel zu sehen.

- Multiplizitäten umfassen nur noch genau ein Intervall
- Attribute werden nur noch als durch Assoziation mit der umgebenden Klasse verbunden betrachtet
- Die Attribute und Operationen treten innerhalb der Klasse in alphabetische Ordnung auf
- Kursivschreibweise ist den abstrakten Elementen vorbehalten und wird nicht mehr für Assoziationsnamen verwand (Bild: Lichterzeugung)
- Parameterspezifikation „return“ für Operationen eingeführt
- Der „Realize“ Stereotyp wird jetzt vollständig durch Lollipop Notation ersetzt (siehe Bild Reizquelle)
- Der Stereotyp „use“ kann in Lollipop Notation mit einem Halbkreis dargestellt werden (siehe Bild Reizquelle)
- Der Begriff Diskriminator für Generalisierungsgründe entfällt, dafür wird die Generalisierungsmenge und optional die Generalisierungseigenschaft angegeben. Obiges Bei-

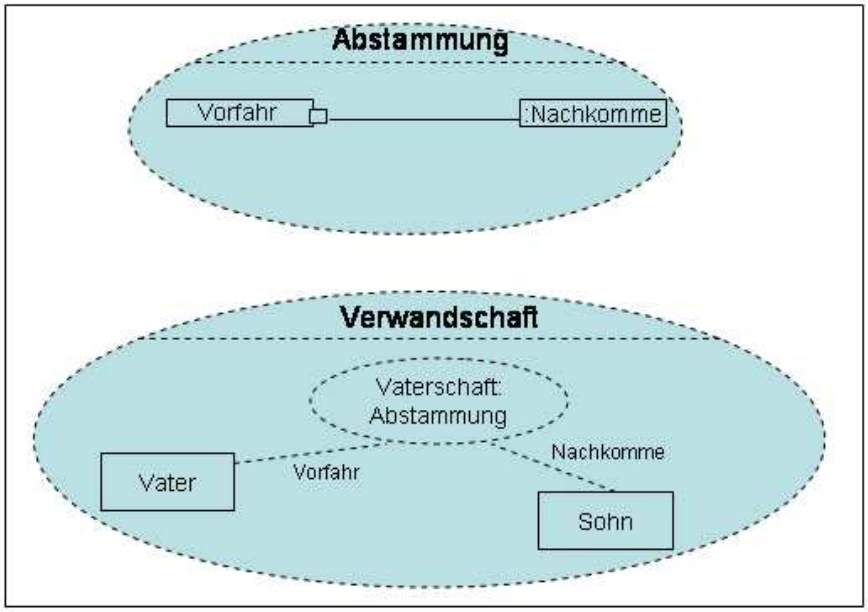
spiel definiert als Menge Aktivierungsart da beide aktiviert werden können und die Eigenschaft ist disjoint (früher Diskriminator) da keine Instanz von Kerze gleichzeitig eine Glühbirne sein kann.

- Kennzeichnungen unveränderlicher Attribute und Assoziationen werden nicht mehr als „frozen“ gekennzeichnet sondern als „readonly“
- Innere Klassen haben keine eigenständige Notation mehr sondern werden als Komposition ausgedrückt

Man sieht also das die Änderungen im Klassendiagramm sind mehr Schönheitsoperationen die auf der ersten Blick nicht unbedingt auffallen. Auf ein Wort zu den Anfangs erwähnten Classifier, die im obigen Bild verwendeten Notationen der Klassen Lichtquelle und Lichtschalter entsprechen der der Reinform eines Classifiers. Auf diese Art kann in allen Diagrammen jedes Notationselement dargestellt werden. Somit ist es auch möglich unsere Lichtquelle und den Lichtschalter direkt in einem Aktivitätsdiagramm zu verwenden als Objektknoten worauf später noch einmal kurz hingewiesen wird.

3.1.2 Das Kompostionsstrukturdiagramm

„Wie sind die einzelnen Architekturkomponenten strukturiert und wie spielen sie zusammen?“



Das Kompositionstrukturdiagramm ist in der UML 2 komplett neu eingeführt wurden. Es zeigt die interne Struktur eines Classifiers (z.B. „Klasse“, “Komponente“) sowie dessen Interaktionsbeziehungen zu anderen Systembestandteilen. Der Kollaborationstyp und als dessen Ausprägung die Kollaboration wurden hierbei als eigenständige Modellelemente eingeführt. Folgende Notationselemente werden in dieser Diagrammart verwand.

- Parts (Vorfahr , Nachfahr)
- Port mit Konnektoren (das kleine Viereck am Vorfahr und die Linie als Konnektor)
- Kollaborationstyp (Abstammung)
- Kollaboration (Vaterschaft)

Im eingehenden Beispiel wird die innere Struktur einer Abstammung und einer Verwandtschaft dargestellt. Zur Ergänzung soll hier angemerkt werden das eigentlich zur Komplettierung der Verwandtschaftsbeziehung auch eine Kollaboration Mutterschaft aus dem Typen Abstammung innerhalb der Verwandtschaft hätte ausgeprägt werden müssen.
Dazu wird ein Kollaborationstyp Abstammung definiert mit den Parts Vorfahr und Nachfahr welche bestimmte „Rollen“ innerhalb der Kollaboration einnehmen. Diese beiden sind über einen Kommunikationskanal dem Konnektor verbunden. Konnektoren können direkt mit Classifiern kommunizieren wie bei Nachkomme oder sie kommunizieren über Ports wie bei Vorfahr. Die Verwendung von Ports dient der Kapselung von Classifiern wobei Ports nichts weiter sind als eine Menge von Schnittstellen und Operationen die der Kollaborationstyp bereitstellt. Zur Vervollständigung sei erwähnt dass auch eine alternative Notation zulässig ist. Bei dieser werden die Rollen außerhalb als Classifier ausgeprägt und mit Konnektoren angebunden an den Kollaborationstyp. Um den Kollaborationstyp Verwandtschaft darzustellen wird Abstammung als Vaterschaft ausgeprägt und mit den Classifiern verbunden. Durch das annotieren von Vorfahr und Nachkomme wird festgelegt welche Rolle der jeweilige Classifier innerhalb des ausgeprägten Kollaborationstyp einnimmt. Die Genauigkeit der Kollaborationstypen wird festgelegt durch die Informationen die zur Erfüllung des Zieles notwendig sind. Denn mehr Informationen als zur Zielerfüllung notwendig werden nicht dargestellt.

3.1.3 Das Komponentendiagramm

„Wie ist mein System strukturiert und wie werden diese Strukturen erzeugt?“

Innerhalb eines Diagramms werden verschiedene Bestandteile eines Systems als Komponenten dargestellt und aufgezeigt wie diese zur Laufzeit organisiert sind. Innerhalb der UML 2 wurde das Diagramm neu gefasst und überarbeitet mit folgenden Änderungen:

- Stereotyp <<implement>> durch <<manifest>> ersetzt
- Stereotypen <<device>> ,<<subsystem>> und <<execution environment>> neu eingeführt
- Stereotyp <<table>> entfällt
- Die alte Komponentennotation entfällt und wird durch Classifier dargestellt
- Eine Komponente ist jetzt eine Sonderform der Klasse
- Artefakte können jetzt praktisch jeden Classifier „manifestieren“

In der UML 2 sollen gängige Markenstandards jetzt besser unterstützt werden. So zum Beispiel .NET oder J2EE.

3.1.4 Das Verteilungsdiagramm

„Wie werden Komponenten des Systems zur Laufzeit wohin verteilt?“

Also zeigt uns dieses Diagramm eine dynamische Zuordnung von Softwarekomponenten auf Hardware-Einheiten. Diese Hardware Einheiten werden als Knoten bezeichnet. Des Weiteren werden die Kommunikationsbeziehungen zwischen den Knoten dargestellt. Mit der Verteilung der Software auf das System ist zum Beispiel die Installation, Konfiguration, Bereitstellung oder Ausführung von Informationseinheiten gemeint. Diese werden auch als Artefakte bezeichnet. Folgende Änderungen haben sich beim Versionswechsel ergeben.

- Gerät, Entwicklungsumgebung, Einsatzspezifikation sind als neue Elemente verfügbar

- Knoten können hinsichtlich ihres Ausführungszeitpunktes feiner spezifiziert werden
- Auch hier ist <<implement>> Stereotyp durch <<manifest>> ersetzt wurden
- Artefakte sind jetzt modellierbar als Implementierung von Packageable Elements

3.2 Die Verhalten –und Interaktionsdiagramme

In diesem Kapitel sollen nun dynamischen Aspekte der Modellierung einmal vorgestellt werden und in welcher Form sie Veränderungen unterzogen wurden. Zuvor soll an dieser Stelle aber noch mal kurz auf das Konzept des Classifiers eingegangen werden und in welcher Form diese Konzept in der dynamischen Modellierung von Interaktionen umgesetzt wurde.
Im Meta Modell der UML ist es möglich jedem Classifier(Klasse, Objekt, Schnittstelle, Komponente…) Verhaltensbestandteile zu zuordnen. 2 Davon sind durch das Meta Modell vordefiniert zum einen die Operation und zum anderen die Empfänger. So ist zum Beispiel ein Verhaltensbestandteil einer Klasse die Operation und dessen Implementierung die Methode ist die zugehörige Verhaltensspezifikation. Das Gesamtverhalten eines Classifiers, zum Beispiel die Änderungen der Attribute einer Klasse, kann als Zustandsautomat dargestellt werden. Nun liegt es nahe alle gemeinsamen Eigenschaften von Verhaltensspezifikationen wie Zustandsautomaten, Interaktionen oder Aktivitäten wieder auf der abstrakten Ebene als einen Classifier zu definieren. Dadurch sind alle Verhaltensspezifikationen:

- Als Klasse darstellbar mit Stereotyp <<activity>> <<statemachine>>
- Spezialisierbar –und generalisierbar
- Hierarchisch zerlegbar
- Als Klassen darstellbar und somit auch instanzierbar

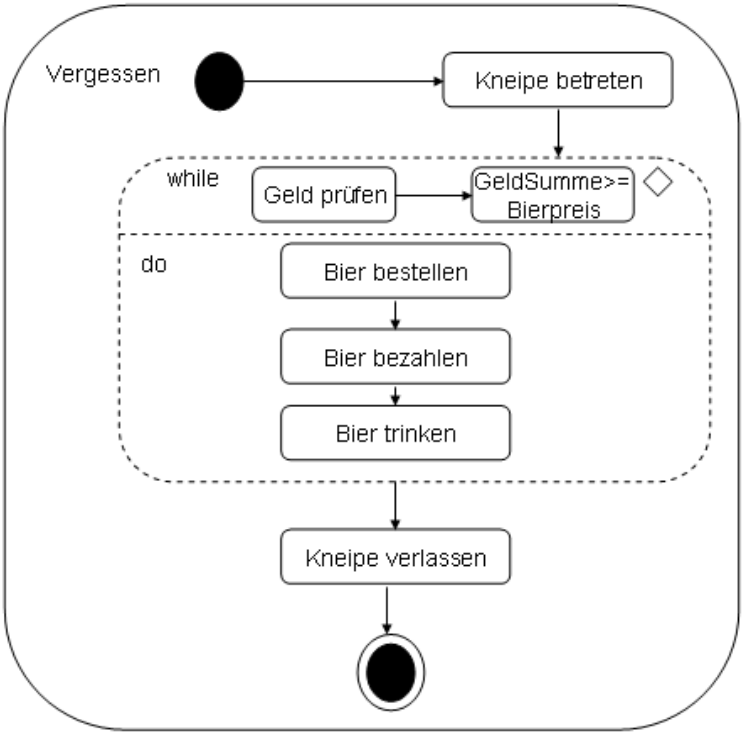
Man kann den Sachverhalt auch von der praktischen Seite her beschreiben und einmal die Interaktionsdiagramme näher betrachten. Jede Operation eines Classifiers kann ich als Sequenzdiagramm, Kommunikationsdiagramm oder auch jegliches andere Dynamische Diagramm beschreiben allerdings mit unterschiedlichen Nach –und Vorteilen. Jede Operation wird also als Verhaltensspezifikation gesehen. Dieser Umstand macht die Einführung des Interaktionsübersichtsdiagramms erst möglich.
Hauptaugenmerk dieses Kapitels soll nun auf den neu eingeführten Timing –und Interaktionsübersichtsdiagrammen liegen sowie dem vielen Änderungen unterzogenen Aktivitätsdiagramm. Die Änderungen in den anderen Diagrammen seien an dieser Stelle nur kurz erwähnt. Das Use-Case-Diagramm und der Zustandsautomat sind nahezu unverändert geblieben genauso wie das Kollaborationsdiagramm außer das es jetzt Kommunikationsdiagramm heißt. Im Sequenzdiagramm haben sich auch wenige Änderungen dafür tief greifende ergeben:

- Anstatt Objekten können jetzt alle zur Interaktion fähigen Partner interagieren
- Interaktionsdiagramme können sich gegenseitig referenzieren
- Alle Kontrollflussmöglichkeiten zur Programmsteuerung sind notierbar
- Die Unterstützung nebenläufiger Programmierung wurde verbessert
- unspezifizierte Nachrichten wurden durch lost –und found Nachrichten ersetzt
- zusätzlich wurden einige Notationselemente zur besseren Strukturierung eingeführt

Das Sequenzdiagramm erlaubt somit eine Modellierung ganzer Verhaltensbereiche in denen Interaktionssequenzen beliebig strukturierbar –und zerlegbar sind.

3.2.1 Das Aktivitätsdiagramm

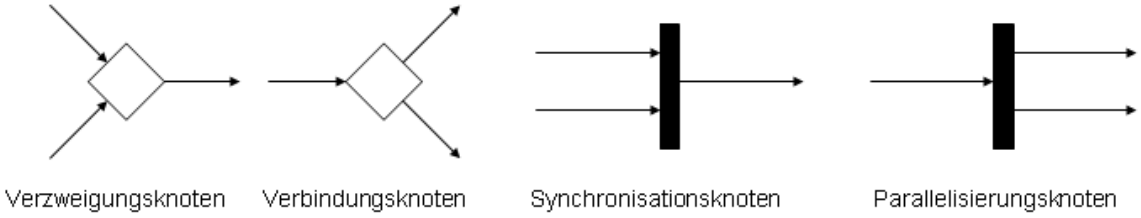
„Wie realisiert mein System ein bestimmtes Verhalten“



Im Aktivitätsdiagramm steht also eine vom System zu bewältigende Aufgabe im Mittelpunkt die es in Einzelschritte zu zerlegen gilt. Es ist das Diagramm was die meisten Änderungen durch den Versionswechsel erfahren hat. Das wichtigste ist das es jetzt keine Sonderform des Zustandsautomaten mehr ist sondern auf einem Token –Konzept basiert ähnlich dem von Petrinetzen. Token kann man sich ungefähr so vorstellen wie den Stab bei einem Staffellauf der anzeigt wo im Lauf wir uns gerade befinden. Durch diese Änderung sind die Diagramme leichter auf Verklemmungsfreiheit testbar. Was sind die weiteren Änderungen?
Für etwas Verwirrung sollten die neuen Bezeichnungen sorgen. Denn jetzt heißen die Diagramme Aktivitäten und die früheren Aktivitäten heißen Aktionen. Dies ist soweit sinnvoll das nun aus einer Aktion neue Aktivitäten heraus aufgerufen werden können oder Use –Cases und Interaktionen. So kann im obigen Beispiel hinter der Aktion „Kneipe Verlassen“ eine neue Aktivität definiert werden. Alle Aktionen können jetzt mit Vor –und Nachbedingungen verknüpft werden. Die Notationen für eine Aktion ist gleich der eines Zustandes. Eine weitere Verbesserung im Bezug auf die Parallelisierung ist durch die Möglichkeit gegeben mehrere Startknoten zu definieren und die parallelen Abläufe müssen nicht mehr zusammengeführt werden. Des Weiteren werden jetzt mehrere Endknoten zugelassen wobei nur genau ein Endknoten das Ende der Aktivität definiert alle anderen definieren nur das Ende eines Ablaufstranges. Die Start –und Endknoten können durch Parameter die in bzw. aus der Aktivität herein oder herausgegeben werden. So könnte im obigen Beispiel der Startknoten durch einen Eingangsparameter „Nüchterner“ und der Endknoten durch einen Ausgangsparameter „Betrunkener“ abgewandelt werden.
An Notationselementen wurden folgende neu eingeführt:

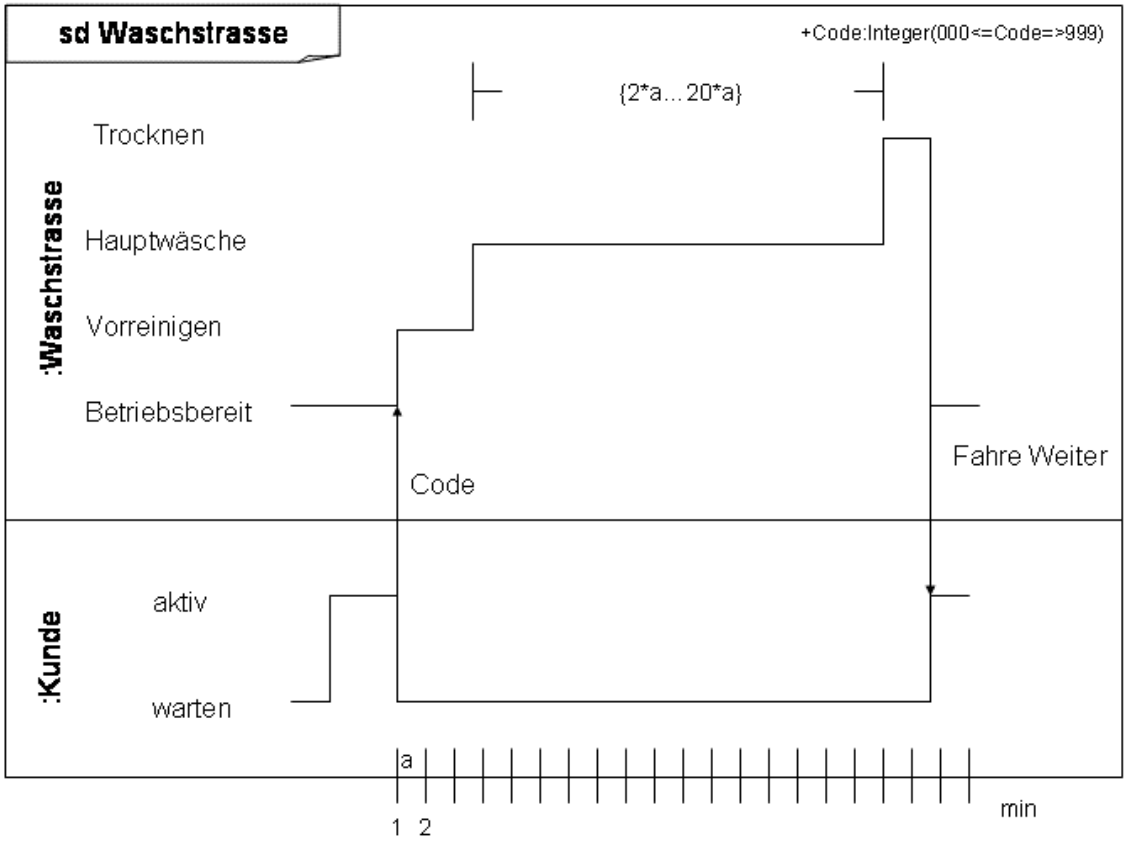
- Strukturierte Knoten
- Mengenverarbeitungsknoten
- Entscheidungsknoten
- Schleifenknoten
- Datenspeicher –und Bufferknoten
- Unterbrechungsbereich
- Parametersatz

Die in voriger Version verwendeten Jumphandler wurden durch Exceptionhandler ersetzt. Im nachfolgenden sind noch einige Kontrollelemente aufgezeigt die verwendet werden um parallele Abläufe sowie Entscheidungen zu modellieren. Sie werden auch im Interaktions-übersichtsdiagramm verwandt.



3.2.2 Das Timing Diagramm

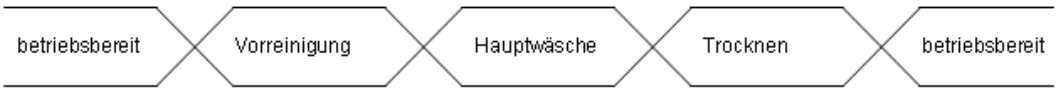
„Wann befinden sich verschiedene Interaktionspartner in welchem Zustand“



Dieses Diagramm wurde komplett neu eingeführt in der UML Version 2. Es wurde aus der Elektrotechnik übernommen wo es gern für digitale Schaltungen eingesetzt wurde. Es kann zur präzisen Analyse und Darstellung des zeitlichen Verhaltens von Classifiern (z.B. Klassen, Akteuren, Komponenten…) verwendet werden. Im Folgenden vielleicht kurz einige Kriterien die berücksichtigt werden sollen wenn Timing Diagramme eingesetzt werden sollen. Das darzustellende System sollte ein reaktives oder stark modulares System sein. Die genauen zeitlichen Übergänge sind von hoher Wichtigkeit und die globalen und lokalen Daten weniger wichtig. Die Interaktion ist sehr einfach gestrickt sodass Nebenläufigkeiten und Kontrollelemente in den Hintergrund treten.
Die folgenden Notationselemente stehen im Timing Diagramm zur Verfügung:

- Lebenslinien
- Nachrichten
- Zeitbedingungen
- Bedingungen, Zustände und Attributwerte von Lebenslinien
- Zeitverlaufslinien
- Wertverlaufslinie

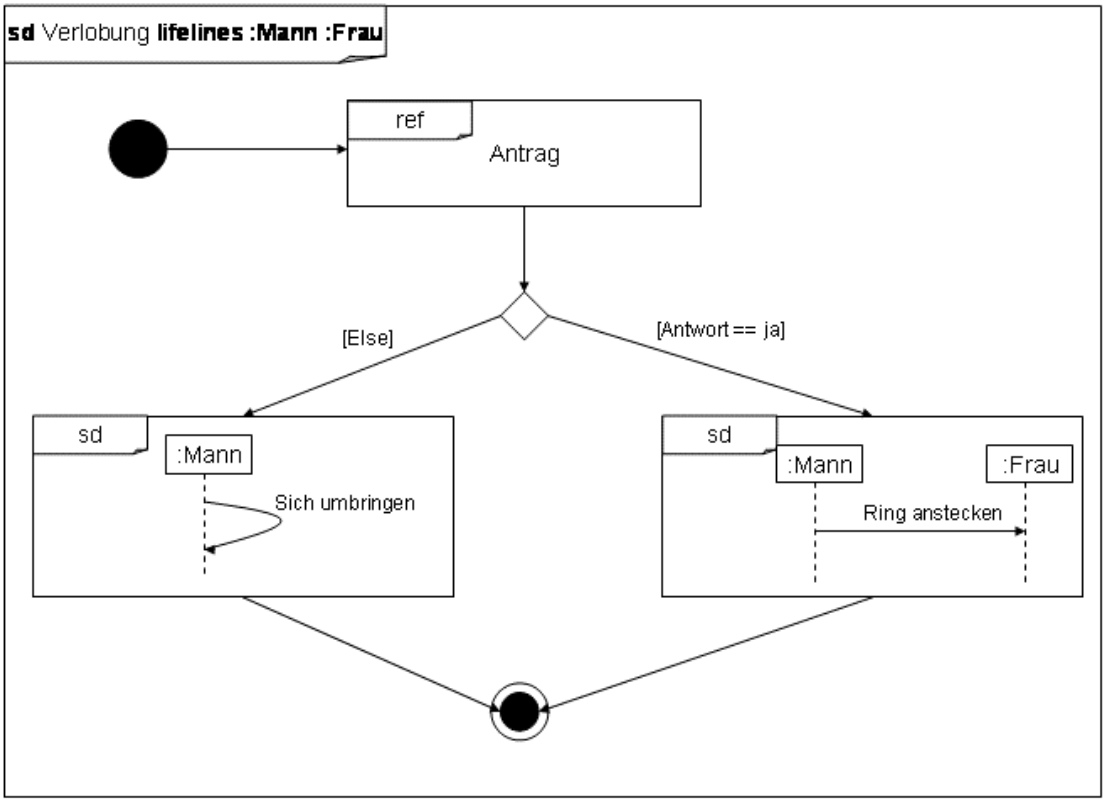
Im eingehenden Beispiel ist ein Timing Diagramm einer Waschstrasse modelliert. Es steht in einem Interaktionsrahmen mit der Bezeichnung „sd Waschstrasse“. Interaktionsrahmen werden auch im den anderen Verhaltensdiagrammen benutzt und irreführender Weise mit einem sd für „Sequenzdiagramm“ beschriftet. In der vertikalen Dimension des Interaktionsrahmens werden die Kommunikationspartner, hier Kunde und Waschstrasse, angetragen dabei stellt jeder breite Streifen die Lebenslinie dar. In der Horizontalen verläuft üblicherweise die Zeitachse die mit einer Zeitskala versehen ist, im Beispiel mit einer Auflösung von einer Minute. Die Linien innerhalb der Lebenslinie(oder des Balkens) ist die Zeitverlaufslinie des Classifiers. Die waagerechten abschnitte bestimmen zeigen an in welchem Zustand sich ein Classifier befindet und die senkrechten Abschnitte zeigen einen Zustandswechsel an. Die Zustände, im Beispiel z.B. warten und aktiv, werden zur linken angezeigt. Zustandswechsel können durch die Angabe einer Nachricht zwischen den Kommunikationspartner erweitert werden, so muss der Kunde beispielsweise den Code der Wäsche angeben die er wünscht. Diese Nachrichten können innerhalb des Diagrams in ihrer Art näher spezifiziert werden wie bei Code geschehen. Also kann eine Wäsche im Beispiel zwischen 2 und 20 Minuten dauern was im Diagramm auch verdeutlicht durch das annotieren des Bereichs über dem Zustand der Hauptwäsche. Zustandsänderungen die nur zögernd von statten gehen können durch eine schräge Zeitverlaufslinie notiert werden. Wenn zu viele unterschiedliche Übergänge stattfinden ist es möglich von der Zeitverlauslinie zur Wertverlauslinie zu wechseln welche die Zustandsänderungen anzeigt wie im nachfolgenden Bild für die Waschstrasse kurz aufgezeigt.



Die Dauer der einzelnen Bereiche würde hier innerhalb des Diagramms für jeden Bereich angegeben werden, wobei ein Zustandswechsel bei der Überschneidung stattfindet.

3.2.3 Das Interaktionsübersichtsdiagramm

„In welcher Reihenfolge und unter welchen Bedingungen finden Interaktionen statt“



Auch das Interaktionsübersichtsdiagramm ist in der UML 2 neu eingeführt und in seinen Grundprinzipien nichts weiter als eine Variante des schon vorgestellten Aktivitätsdiagramms. Allerdings werden hier keine abfolgen von Aktionen dargestellt sondern Abfolgen von Interaktionen. In Einführung der dynamischen Diagramme wurde ja schon erwähnt das alle abstrakt gesehen Verhaltensspezifikationen darstellen. Ganz Allgemein kann ich also dadurch verschiedene schon modellierte Aspekte meines Systems in einem großen Zusammenhang darstellen ohne den Überblick zu verlieren. Dabei ist gleichermaßen ein Top Down Ansatz wie ein Bottom Up Ansatz möglich. Das Interaktionsdiagramm ermöglicht es Referenzen auf schon erstellte dynamische Diagramme sowie komplett neue sozusagen „inline“ zu modellieren. Hierbei stehen die gängigsten Notationselemente des Aktivitätsdiagramms wie zum Beispiel Verzweigungen zur Verfügung. Objektknoten und Aktionen entfallen. Anstelle der Aktionen werden einfach Interaktionsrahmen modelliert. Man sollte allerdings auf einem übermäßigen Einsatz der Kontrollelemente verzichten da dies die Übersichtlichkeit welche das eigentliche Ziel ist aus dem Auge verliert.
Im obigen einfachen Beispiel eines Heiratsantrages ist zum Beispiel dessen Ablauf im gesamten dargestellt. Der Interaktionsrahmen Antrag selbst verweist allerdings auf ein anderes Diagramm in dem die konkrete Ausmodellierung vorgenommen wurde. Es geht allerdings nicht daraus hervor was für ein Diagramm es ist da es in dieser Sicht auch nicht relevant ist. Nach dem Antrag kommt ein Verzweigungsknoten mit der Entscheidung ja oder nein wie sie aus dem Aktivitätendiagramm bekannt sind. Die beiden nächsten Interaktionsrahmen modellieren dann direkt im Interaktionsübersichtsdiagramm 2 Sequenzdiagramme. Nach Ablauf dieser ist das Interaktionsdiagramm bei dem Endknoten beendet.