UML – Teil 1

Die Unified Modelling Language, eine Standard-Notation für System-Modellierung, findet immer verbreiteteren Einsatz bei der objektorientierten Analyse und hilft, Probleme effizient zu lösen.

Thomas Obermayer

Ein Vogelhäuschen zu bauen, ist ein relativ triviales Problem. Man benötigt ein paar Bretter, eine Säge, Nägel und ein bisschen Kreativität. Die Phase der Planung nimmt maximal eine Viertelstunde in Anspruch. Man kann praktisch die Idee direkt in ein fertiges Produkt umsetzen.

Einen Wolkenkratzer zu bauen, ist schon etwas schwieriger. Mit einem Architekten analysiert man die Anforderungen an das Haus. Dann entsteht ein Plan, der eine Vorstellung verschaffen soll. Der Plan ist so ausgelegt, dass man ihn ohne große Worte verstehen und deuten kann. Der Bauherr interpretiert diesen Plan und setzt ihn in eine Projektbeschreibung um. Erdarbeiter, Maurer, Tischler, Dachdecker, Elektriker und Installateure werden entsprechend ihrer Fähigkeiten eingeteilt. Immer sind Pläne und Diagramme im Spiel.

Viele Software-Entwickler möchten Wolkenkratzer bauen und verhalten sich dabei, als sollte ein Vogelhaus entstehen. Der Analysephase wird in Software-Projekten eine viel zu unbedeutende Rolle zugemessen. Professionelle Produkte können aber nur dann effizient entwickelt werden, wenn man entsprechend analysiert, designed und vor allem dokumentiert.

Ein nützliches Mittel hierfür stellt die UML (*„Unified Modelling Language“*) dar. Es handelt sich dabei um eine grafische Notation zur (objektorientierten) Modellierung von Systemen. Sie ist eine weit verbreitete Variante und hat sich allgemein durchgesetzt.

Dieser Artikel soll die UML und ihre Diagramme kurz vorstellen, damit Sie lernen, Modelle richtig zu interpretieren. Kenntnisse in der objektorientierten Systementwicklung werden vorausgesetzt.

Die Diagramme

Die UML schlägt mehrere Diagramme für die Darstellung der verschiedenen Ansichten auf ein System vor. Man könnte dies mit den Bauplänen für ein Haus vergleichen (Grundriss, Elektroinstallations-Plan, Finanzierungsplan, etc). Jedes Diagramm ist für eine bestimmte Sicht besser geeignet als ein anderes.

Insgesamt umfasst die UML neun Diagramme:

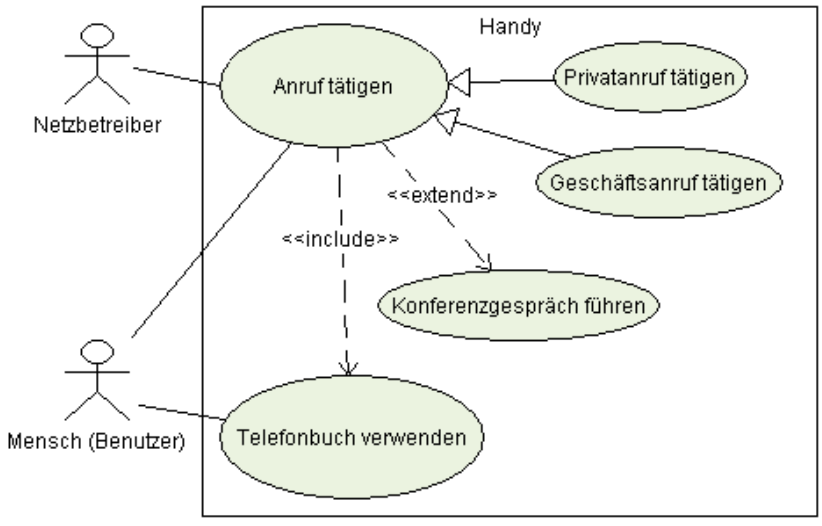
- Use-Case-Diagramme
- Klassen- und Objektdiagramme
- Interaktionsdiagramme (bestehend aus Sequenz- und Kollaborationsdiagrammen)
- Package-Diagramme
- Zustandsdiagramme
- Aktivitätsdiagramme
- Implementierungsdiagramme

Jedes Diagramm hat spezielle Stilelemente und Beschriftungen. Hinzu kommen Mechanismen, die für alle Diagramme anwendbar sind.

Insgesamt ist die UML sehr umfangreich, erlaubt aber eine Darstellung in unterschiedlichen Präzisionsgraden. Man sollte nicht versuchen, ein System gleich bis ins Detail zu modellieren. Die objektorientierte Analyse versteht sich als iterativer Prozess (man beginnt mit einer sehr abstrakten Beschreibung und verfeinert das Modell schrittweise).

Das Use-Case-Diagramm (auch “Anwendungsfalldiagramm“)

Das Use-Case-Diagramm dient dazu, die typischen Anwendungsfälle eines Systems zu beschreiben. Zusätzlich wird der Einfluss eines externen Akteurs auf das System (und die Verbindung zu den jeweiligen Anwendungsfällen) veranschaulicht. Ein Akteur muss nicht unbedingt eine Person sein; auch eine Firma, ein Tier oder ein anderes System wäre möglich.



Use-Case-Diagramm: System („Handy“), Use-Cases („Anruf tätigen“, „Telefonbuch verwenden“, etc), Akteure („Benutzer“, „Netzbetreiber“)

Ein Use-Case (Anwendungsfall) ist eine typische Handlung in einem System, an der ein externer Akteur beteiligt ist, z. B. “Anruf tätigen”. Use-Cases werden als Ellipsen eingezeichnet, wobei die Beschreibung beliebig umfangreich sein kann. Verbindungen zwischen Anwendungsfällen und Akteuren werden durch Linien hergestellt.

Auch Verbindungen von Use-Cases zu anderen Use-Cases sind möglich. Die wichtigsten sind **<<extend>>**, **<<include>>** und die Generalisierung (dreieckiges Pfeilchen).

Die **<<extend>>**-Abhängigkeit zeigt, dass ein optionaler Teil eines Anwendungsfalles ausgelagert wird. Ein optionaler Teil von „Anruf tätigen“ wäre z.B. „Konferenzgespräch führen“.

Um einen obligatorischen Teil auszulagern, der in mehreren Use-Cases vorkommt, bedient man sich der Generalisierungs-Beziehung (Vererbung). Achtung: Wenn ein Use-Case einen anderen generalisiert, dann muss die „ist vom Typ“-Beziehung erfüllt werden, z.B. „der Anwendungsfall ‚Privatanruf tätigen‘ ist vom Typ ‚Anruf tätigen‘“ – der Use-Case „Privatanruf tätigen“ verwendet „Anruf tätigen“ immer – daher wird Generalisierung und keine **<<extend>>**-Beziehung verwendet. Auch die Generalisierung von Akteuren ist zulässig. Verwechseln Sie als JAVA-Programmierer die **<<extend>>**-Beziehung nicht mit Generalisierung!

Möchte man einen Use-Case, der eine Teilmenge mehrerer Use-Cases ist, auslagern (um Redundanzen zu vermeiden), so symbolisiert man dies durch eine **<<include>>**-Abhängigkeit.

Sie haben die Möglichkeit, einzelne Use-Cases durch weitere Use-Case-Diagramme zu verfeinern. Dies empfiehlt sich vor allem bei komplexeren Systemen, damit die Diagramme übersichtlich gehalten werden.

Klassen und Objekte

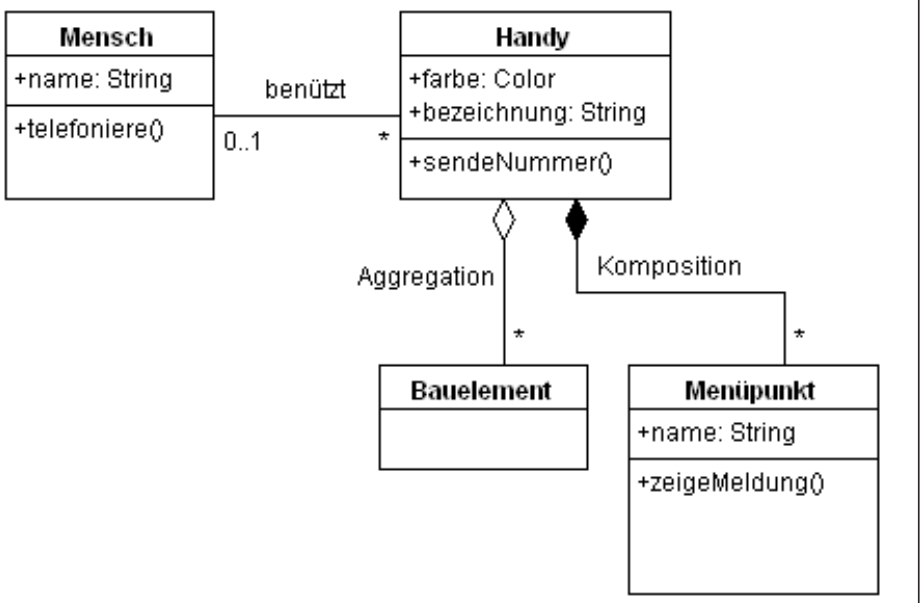
Um die Struktur eines Systems zu analysieren, sind Klassen und Objekte wichtige Elemente.

Man analysiert, aus welchen Objekten (Konzepte, Abstraktionen oder Gegenstände mit klarer Abgrenzung und präziser Bedeutung, z. B. "das blaue Handy" und „das rote Handy") ein System besteht. Dann filtert man alle Objekte heraus, die gleich aufgebaut sind (z.B. hat jedes Handy eine Farbe und einen Preis, kann eine Nummer senden, etc) und bildet eine Klasse (Gruppe von Objekten mit ähnlichen Eigenschaften, z.B. "Handy"). Die Klasse dient als Vorlage für weitere Objekte dieses Typs. Man nennt die Erstellung eines Objektes (bzw. „Instanz") aufgrund einer Klasse „Instanzierung".

Es gibt verschiedene Möglichkeiten, Klassen in einem System zu finden. Die UML enthält keine Aussage über entsprechende Techniken, sie ist nur eine Notation.

Das Klassendiagramm

Das Klassendiagramm ist das häufigste Diagramm der UML und stellt den zentralen Punkt eines objektorientierten Modells dar. In ihm werden sämtliche Klassen des Modells und die Beziehungen der Klassen zueinander angezeigt.



Klassendiagramm: Klassen („Mensch", „Handy", „Bauelemente", „Menüpunkt") und Beziehungen

Die Klasse (z.B. „Mensch") ist das zentrale Element des Diagramms. Sie wird durch ein Rechteck mit Bezeichnung (fett) dargestellt. Optional kann sie aus zwei zusätzlichen Bereichen bestehen: einem Bereich für die Attribute (z.B. „name") und einem Bereich für die Operationen (z.B. „telefoniere()"). Die Sichtbarkeit der Attribute und Operationen kann wahlweise durch ein vorangestelltes „+" (public), ein „-" (private) oder ein „#" (protected) angezeigt werden.

Beziehungen zwischen Klassen werden durch Linien dargestellt. Sie können einen Namen (z.B. „benützt") haben, um Klarheit zu schaffen. Weiters wird oft die Multiplizität (in der Datenmodellierung auch „Kardinalität" genannt) angegeben („0..1", „*", etc). Beispielsweise möchte man durch die Multiplizität zeigen, dass ein Mensch mehrere Handies benützen kann und ein Handy entweder keinen oder einen Benützer hat (vgl. mit der Abbildung).

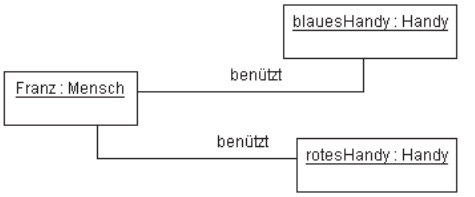
Spezielle Formen der Beziehung sind die Aggregation („besteht aus"-Beziehung) und die Komposition („besteht aus"-Beziehung mit Existenzabhängigkeit). Mit der Aggregation zeigt man an, dass ein Teil aus mehreren kleineren Teilen besteht (z.B. ein Handy besteht aus mehreren Bauelementen). Handelt es sich um eine Aggregations-Beziehung, bei der die Existenz einer unteren Klasse (z.B. „Menüpunkt") von der Existenz der oberen Klasse (z.B. „Handy") abhängt, so bedient man sich der „Komposition" (schwarz ausgefüllte Raute). In der Abbildung wird durch Komposition angezeigt, dass ein „Menüpunkt" nur dann existiert, wenn es ein „Handy" gibt.

Die Generalisierung (Vererbung) von Klassen kennzeichnet man, wie im Use-Case-Diagramm, durch einen Pfeil mit dreieckiger, nicht ausgefüllter Spitze. Der Pfeil geht von der spezialisierten Klasse aus und zeigt auf die Superklasse (vgl. mit Abbildung „Use-Case-Diagramm").

Das Objektdiagramm

Um einen konkreten Fall veranschaulichen zu können, benützt man das Objektdiagramm. In ihm werden Objekte (Instanzen), also konkrete Repräsentanten der Klassen eingezeichnet (z.B. 2 Objekte vom Typ „Handy", ein Objekt vom Typ „Mensch").

Objekte werden durch Rechtecke mit Bezeichnung dargestellt. Die Bezeichnung



Objektdiagramm: Instanzen („Franz", „blauesHandy", „rotesHandy"), Beziehungen („benützt")

setzt sich aus dem Objektnamen, einem Doppelpunkt und der zugehörigen Klasse zusammen (z.B. „Franz : Mensch"). Die Bezeichnung ist unterstrichen, um eine schnelle Unterscheidung zum Klassendiagramm zu ermöglichen.

Das Klassendiagramm dient als Vorlage für das Objektdiagramm. Man entnimmt dem Klassendiagramm, von welchem Typ (von welcher Klasse) eingezeichnete Objekte sein können. Anhand der Multiplizität im Klassendiagramm sieht man, wie viele Objekte daraus entstehen (z.B. eine „1 zu 2"-Beziehung zwischen „Mensch" und „Handy" im Klassendiagramm führt zu einem Objekt vom Typ „Mensch" und zu zwei Objekten vom Typ „Handy" im Objektdiagramm). Im Fall einer „1 zu *"-Beziehung ist es nicht möglich, im Objektdiagramm einen konkreten Fall darzustellen. Daher zeichnet man so viele Objekte ein, wie man als sinnvoll erachtet. Im Objektdiagramm wird keine Multiplizität angegeben.

Beziehungen im Objektdiagramm werden als Instanzen der Beziehungen im Klassendiagramm aufgefasst, d.h. eine „benützt"-Beziehung mit der Multiplizität „1 zu 2" im Klassendiagramm wird zu zwei „benützt"-Beziehungen im Objektdiagramm (vgl. mit den jeweiligen Abbildungen).

Da die Kommunikation zwischen Objekten im Objektdiagramm nicht dargestellt wird, findet man es in Modellen eher selten. Um das Zusammenwirken von Instanzen darzustellen, sieht die UML sog. „Interaktionsdiagramme" vor, die, gemeinsam mit dem Package-Diagramm, in der nächsten Ausgabe von **PCNEWS** vorgestellt werden.

Die Darstellungen in diesem Beitrag wurden mit ObjectDomain 2.5 (http://www.objectdomain.com/) erstellt. Dieses Programm läuft auf allen Plattformen und erlaubt sowohl Forward- als auch Reverse-Engineering. Kosten: $ 495,-.

Thomas Obermayer thomas@obermayer-it.at