UML - Teil 3

Im dritten Teil der PCNEWS-Artikelreihe werden Diagramme erläutert, die sowohl in der objektorientierten Modellierung als auch im Software-Projektmanagement und während des Entwicklungsprozesses von Hardware-Komponenten eingesetzt werden können.

Thomas Obermayer

Bislang stellten wir Ihnen UML-Diagramme vor, die zur Darstellung von statischen und dynamischen Eigenschaften verschiedener Systeme dienen. Klassen-, Objekt- und Paketdiagramme beschreiben die statische Sicht auf Klassen, ihre Repräsentanten (Objekte) und die Aufteilung der Klassen in einzelne Namensräume (Pakete). Mit Kollaborations- und Sequenzdiagrammen wird es dem Entwickler möglich, Interaktionen zwischen Objekten zu modellieren ***(vgl. „UML - Teil 1 und 2“)***.

Die nun erläuterten Diagrammtypen erlauben zusätzlich die Beschreibung von Prozessen, die Darstellung von Zustandsänderungen in “Automaten” und die Angabe von Informationen bezüglich der Implementierung.

Das Aktivitätsdiagramm

Aktivitätsdiagramme dienen dazu, die dynamischen Aspekte von Systemen zu modellieren. Im Wesentlichen handelt es sich dabei um Flussdiagramme, die den Kontrollfluss von Aktivität zu Aktivität zeigen.

Im Gegensatz zu Sequenz- oder Kollaborationsdiagrammen steht hier nicht der Kontrollfluss zwischen Objekten im Vordergrund! Aktivitätsdiagramme können auch verwendet werden, um die dynamischen Gesichtspunkte einer ganzen Gruppe von Objekten oder den Ablauf einer Operation zu beschreiben. Es lässt sich so zum Beispiel die Chronologie eines gesamten Projektes grafisch darstellen.

Aktivitätsdiagramme setzen sich aus folgenden Grundelementen zusammen:

- Aktivitäten,
- Zustandsübergänge („Transitionen“),
- optionalen Synchronisationslinien,
- optionalen Verzweigungen,
- optionalen Schwimmbahnen (“Swimlanes”) und
- einem Anfangs- und einem Endzustand.

Aktivitäten sind Zustände, in denen Vorgänge ablaufen. Sie werden durch abgerundete Rechtecke mit einer Beschreibung dargestellt.

Zustandsübergänge („Transitionen“) erfolgen automatisch am Ende der Aktivitäten und werden durch Pfeile dargestellt.

Synchronisationslinien benötigt man, um parallele („nebenläufige“) Aktivitäten darzustellen. Sie schalten, wenn alle Eingangstransitionen (alle auf sie weisenden Zustandsübergänge) vorhanden sind. Zusammenführungen von separaten Aktivitäten zu einer lassen sich analog modellieren. Synchronisationslinien werden durch schwarze Balken dargestellt.

Wie aus herkömmlichen *Flow-Chart*-Diagrammen bekannt, gibt es auch in UML-Aktivitätsdiagrammen die Möglichkeit, bedingte Verzweigungen zu modellieren. Eine Verzweigung wird als Raute dargestellt. Sie kann einen Eingang und zwei oder mehr Ausgänge haben. An jeden Ausgang schreibt man einen boolschen Ausdruck, der nur ausgewertet wird, wenn die Verzweigung erreicht wird. Man könnte derartige Verzweigungen später z.B. als ‘if ... then ... else’ - Instruktionen implementieren.

“Swimlanes” werden verwendet, um die Aktivitätszustände eines Diagramms in Gruppen zusammenzufassen, wobei jede Gruppe eine Organisationseinheit darstellt, die für diese Aktivitäten verantwortlich ist. Man nennt jede solche Gruppe „Verantwortlichkeitsbereich“ oder „Schwimmbahn“ *(„Swimlane“)*. Jeder Verantwortlichkeitsbereich erhält einen Namen. Schwimmbahnen werden durch vertikale Linien getrennt; wobei man pro Verantwortlichkeitsbereich eine eigene Spalte verwendet ***(vgl. mit der Abbildung)***.

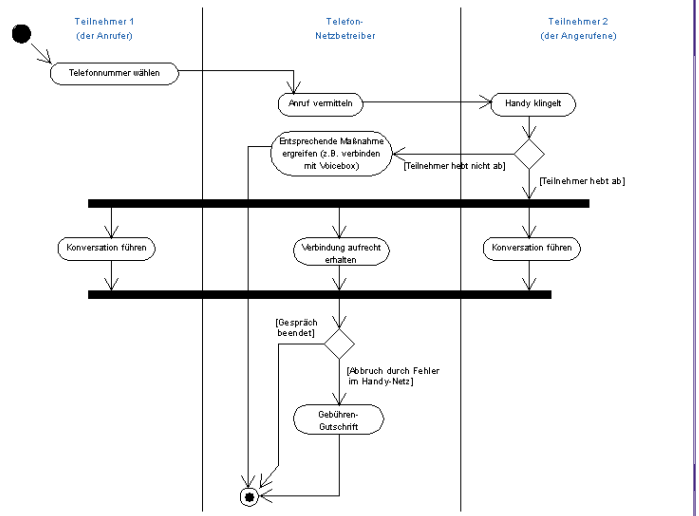
Anfangs- und Endzustand werden durch entsprechende Kreise dargestellt (ein voller Kreis für den Anfangszustand und ein voller Kreis innerhalb eines ungefüllten Kreises für den Endzustand).

Das Zustandsdiagramm

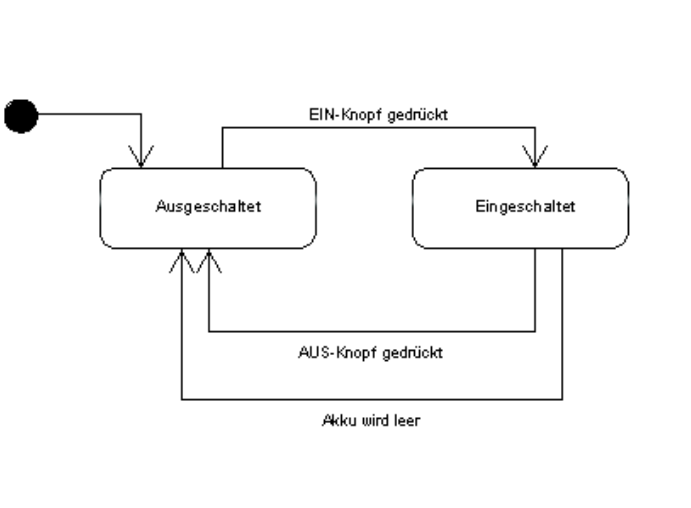
Jedes Objekt befindet sich zu jeder Zeit in einem bestimmten Zustand (z.B. ist ein Handy entweder ein- oder ausgeschaltet). Derartige Zustände können bei der Implementierung durch Variablen gespeichert werden.

Für manche Aufgabenstellungen ist es sinnvoll, das dynamische Verhalten einer Klasse durch die Darstellung ihrer möglichen Zustände anzugeben. Zusätzlich definiert man, durch welche Einflüsse bzw. Vorgänge die Klasse von einem Zustand in einen anderen gelangt.

Aktivitätsdiagramm „Ein Telefongespräch“: Aktivitäten (z.B. „Telefonnummer wählen“), Transitionen, Verzweigungen (Rauten), Synchronisationslinien (schwarze Balken), Schwimmbahnen (z.B. „Teilnehmer 1 (der Anrufer)“)



Zustandsdiagramm „Handy als Automat“: Zustände (z.B. „Ausgeschaltet“), Transitionen (z.B. „Akku wird leer“)



Durch die Angabe von Zuständen und Zustandsübergängen ist es möglich, das gesamte dynamische Verhalten der Klasse zu modellieren. Hierbei wird der zeitliche Fluss der Vorgänge außer Acht gelassen.

Zustandsdiagramme sehen ähnlich aus wie Aktivitätsdiagramme. Zustände werden durch Rechtecke mit abgerundeten Ecken gezeichnet (die Abrundungen sind kleiner als jene in den Aktivitätsdiagrammen).

Durch das Eintreffen von Ereignissen kann ein anderer Zustand erreicht werden, was durch die Pfeile angedeutet wird, die eine Beschriftung für das auslösende Ereignis haben. Die Pfeile stellen die Zustandsübergänge („Transitionen") dar.

Wie im Aktivitätsdiagramm, kann man auch im Zustandsdiagramm Anfangs- und Endzustand durch entsprechende Kreise einzeichnen.

In der Praxis lassen sich Systeme mit der umgangssprachlichen Bezeichnung „Automat" (z.B. „Geldautomat", „Spielautomat", etc.) am besten durch Zustandsdiagramme beschreiben. Diese Möglichkeit findet vor allem in der Entwicklung von elektronischen Hardware-Komponenten Anwendung.

Das Komponentendiagramm

Die UML stellt zwei Diagrammtypen zur Beschreibung der tatsächlichen Implementierung zur Verfügung, die gesamt „Implementierungsdiagramme" genannt werden. Diese werden in Komponentendiagramme und *Deployment*-Diagramme eingeteilt.

Damit bei späterer Implementierung der Softwarelösung Compiler- und Laufzeitabhängigkeiten klar sind, werden die Zusammenhänge der einzelnen Komponenten der späteren Softwarelösung in einem Komponentendiagramm dargestellt.

Komponenten werden als Rechtecke dargestellt, die den Namen und eventuell den Typ der jeweiligen Komponente enthalten. Am linken Rand jeder Komponente befinden sich zwei kleine Rechtecke.

Eine Komponente kann weitere Elemente, wie Objekte, Komponenten oder Knoten enthalten.

Die Abhängigkeiten zwischen den einzelnen Komponenten werden durch gestrichelte Pfeile symbolisiert.

Anhand des so gezeichneten Diagramms lässt sich die spätere Kompilierreihenfolge erkennen (die Abbildung zeigt, dass die Komponente „Handy-Applikation.java" die Komponenten „GUI.java" und „BIOS.java" benützt; daher müssen die beiden zuletzt genannten Dateien zuerst kompiliert werden).

Das Deployment-Diagramm

Zur Darstellung der Hardware-Plattform werden Deployment-Diagramme verwendet. Man versucht, einen kompakten Überblick über die Zielplattform zu schaffen.

Verarbeitungs- bzw. Hardwareeinheiten werden durch sogenannte Knoten dargestellt. Diese sind als beschriftete Quader gezeichnet. Wie in den Interaktionsdiagrammen, zeichnet man Objekte (z.B. „S35") als Instanzen von Klassen (z.B. „Handy") ein.

Unter den Knoten existieren Verbindungen. Dabei handelt es sich um die physikalischen Kommunikationspfade, die als Linien eingezeichnet werden. Durch Beschriftungen kann die Art der Pfade angegeben werden.

ArgoUML

Die Grafiken zu diesem Beitrag wurden mit ArgoUML 0.8.1 erstellt. Dabei handelt es sich um ein freies Open-Source-Werkzeug, das in JAVA entwickelt wurde und somit auf allen Plattformen läuft. Nähere Informationen bietet die Website http://www.argouml.org.

Literaturhinweise

Die in dieser Artikelreihe erklärten Elemente sollten reichen, um einen Einstieg in die Objektorientierte Systementwicklung mittels UML zu erlangen. Die *Unified Modeling Language* kennt jedoch weit mehr Elemente und ist eine komplexe Sprache, die eine geschlossene Behandlung von computerorientierten Problemstellungen ermöglicht.

Dem interessierten Leser sei folgende Literatur zur Vertiefung empfohlen:

- Grady Booch, Jim Rumbaugh, Ivar Jacobson, Das UML-Handbuch, Addison-Wesley Verlag 1999
- Perdita Stevens, Rob Pooley, UML - Softwareentwicklung mit Objekten und Komponenten, Addisson-Wesley Verlag 1999
- Jim Rumbaugh, Objektorientiertes Modellieren und Entwerfen, Hanser Verlag 1993
- Ivar Jacobson, Object-Oriented Software Engineering, A Use Case driven Approach, Addison-Wesley Verlag 1992
- UML Unified Modeling Language, Version 1.0 und 1.1 (http://www.uml.org)
- Glossar für das Themengebiet UML (http://www.oose.de/glossar)

Softwarehinweise

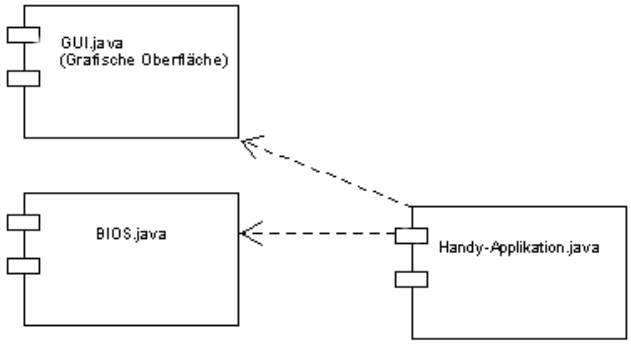
Um den Umgang mit Klassen und Objekten in UML zu erlernen, ist es hilfreich, anfangs die Diagramme per Hand auf Papier zu zeichnen. Man erkennt schnell die häufigsten Probleme und sieht, dass Modellierung und Programmierung einander abwechseln und eine Lösung nur iterativ gefunden werden kann.

Später kann man sich eines CASE-Tools bedienen, mit dem man UML-Diagramme zeichnet, die automatisch in entsprechenden Programmcode übersetzt werden.

Folgende Produkte sind hierfür gut geeignet:

- Rational Rose (http://www.rational.com)
- Together (http://www.togethersoft.com)
- Object Domain (http://www.objectdomain.com)
- ArgoUML (http://www.argouml.org)

Komponentendiagramm „Handy-Software": Komponenten (z.B. „GUI.java"), Abhängigkeiten



Deployment-Diagramm „Handy-Kommunikation": Knoten (z.B. „S35: Handy"), Verbindungen (z.B. „FUNK")