mehr zum thema:
www.uml-glasklar.de

von mario jeckle, chris rupp, jürgen hahn, barbara zengler und stefan queins

UML 2.0: EVOLUTION ODER DEGENERATION?

Der UML-Standard in der Version 2 steht kurz vor seiner endgültigen Verabschiedung durch die OMG. Damit entsteht eine vollständig überarbeitete UML mit neuen Diagrammen, Modellierungselementen und einem deutlich gestrafften Metamodell. Die Sprache wurde semantisch präzisiert und die Diagramme sind näher an die Ausführbarkeit gerückt. Hierbei wurde insbesondere in die Verhaltensdiagramme viel Arbeit investiert. Der Artikel verschafft Ihnen einen Überblick über die erfolgten Änderungen und diskutiert die Frage, ob und für wen sich ein Umstieg auf die UML 2 lohnt.

Die UML ist heute das Mittel der Wahl, wenn es darum geht Softwaresystem zu analysieren und zu entwerfen. In vergleichsweise kurzer Zeit hat sie sich vom kleinsten gemeinsamen Nenner dreier Notationen (Booch, OMT und OOSE), der UML 0.9 von Booch, Rumbaugh und Jacobson, zum De-facto-Standard entwickelt. Ist die UML (1.x) so gut oder waren die Konkurrenznotationen einfach nur so unbrauchbar? Und wenn die UML so gut ist, wieso brauchen wir dann eine neue, vollständig überarbeitete Version?

Nach langer Kontinuität des durch die Object Management Group (OMG) betriebenen Standardisierungsprozesses, der als Ergebnis der Kärrnerarbeit nahezu jährlich evolutionäre Punktversionen der UML produzierte, unternimmt das Industriegremium mit der fast abgeschlossenen Fassung der UML 2.0 einen zentralen Versionssprung. Während sich die Unterschiede zwischen den jeweiligen Vorgängern in vergleichsweise eng umrissenen, überschaubaren Grenzen hielten und zumeist lediglich für Spezialanwender des direkt modifizierten Konstrukts, Berater sowie Werkzeug- und Methodenautoren von Interesse waren, enthält Version 2 der UML eine Reihe von Neuerungen, die auch für den Praktiker sowie für Architekten und Softwareentwickler relevant sind.

Insgesamt betrachtet versucht die UML 2.0 die in den vergangenen rund fünf Jahren Breiteneinsatz der Sprache gewonnenen Anwendungserfahrungen der Modellierer gesammelt aufzugreifen und damit gleichzeitig den inzwischen erhobenen Erweiterungswünschen Rechnung zu tragen. Die UML 2.0 bildet damit bereits im grundlegenden Ansatz eine neue Version der Modellierungssprache, die sich deutlich von korrigierenden Änderungen der verschiedenen Ausprägungen der ersten Standardgeneration abhebt.

Im Detail trachtet die Neufassung danach Änderungen an drei zentralen Punkten der Sprachstruktur einzuführen. So soll gleichzeitig die semantische Präzision der angebotenen grafischen Primitive gesteigert werden, die angebotenen Diagrammtypen (siehe Abb. 1) so ergänzt werden, dass ihre direkte Ausführbarkeit in greifbare Nähe rückt, und dennoch die Übersichtlichkeit der Spezifikation und damit der Modellierungssprache selbst gesteigert werden.

Die unnötige Komplexität der UML-1.x-Versionen und des sie beschreibenden normativen Dokuments stellt zweifelsfrei die am häufigsten geäußerte Kritik der Anwendergemeinde dar, betrifft sie doch direkt den Modellierer, da sie dem schnellen Erlernen der UML sowie ihrer effektiven Nutzung in der Praxis im Wege steht. Um diesem Komplexitätsübel abzuhelfen, fällt die Neufassung – trotz des nachfolgend beschriebenen Mächtigkeitszuwachses – quantitativ deutlich schlanker als ihre Vorgänger aus. Dies konnte durch vielfache Abstraktion der angebotenen Konzepte und die Wiederverwendung der so entstehenden verallgemeinerten Definitionen erfolgen. Eines dieser Konzepte (und zentrales Konzept in der UML überhaupt) ist das des *Classifiers*, welches das objektorientierte Grundkonzept der „Klasse" weiter abstrahiert. Ein *Classifier* ist kein Modellierungselement, sondern ein gedankliches Konstrukt mit gewissen Eigenschaften zur Klassifizierung. Beispiele für *Classifier* sind Use-Case, Zustandsautomat, Akteur und die Klasse selbst. Anzumerken ist jedoch, dass dieser durch Abstraktion erreichte quantitative Gewinn an Übersichtlichkeit durch einen Verlust an intuitiver Zugänglichkeit erkauft wird und sich die offiziellen UML-2.0-Dokumente der OMG schlechterdings kaum für den Einstieg in die Modellierungssprache eignen.

Jenseits der erhobenen Anwenderforderungen birgt auch der Markt einige Gründe für die grundlegende Überarbeitung der UML. Seit der Ur-Version der

die autoren

Mario Jeckle (E-Mail: mario@jeckle.de) ist Professor für Software Engineering an der Fachhochschule Furtwangen. Im Rahmen seiner Tätigkeit in der OMG ist er Koautor der UML 2.0 und Mitinitiator des Austauschformats XML Metadata Interchange.

Chris Rupp (E-Mail: chris.rupp@sophist.de) ist Geschäftsführerin der SOPHIST GROUP und liefert immer wieder wichtige Impulse für die Bereich Requirements Engineering und Objektorientierung.

Jürgen Hahn (E-Mail: juergen.hahn@sophist.de) ist Trainer und Berater bei der SOPHIST GROUP und vermittelt seit mehreren Jahren UML-Notation und OO-Methoden.

Barbara Zengler (E-Mail: bz@barbara-zengler.de) ist Gründerin der Firma Thebit mit den Arbeitsschwerpunkten Modellierung service-orientierter Architekturen und Sicherheitsaspekten web-services-basierter Kommunikation.

Dr. Stefan Queins (E-Mail: stefan.queins@sophist.de) setzt als Berater bei SOPHIST objektorientierte Methoden und Notationen in den Bereichen der Systemanalyse und Architektur von technisch orientierten Systemen ein.

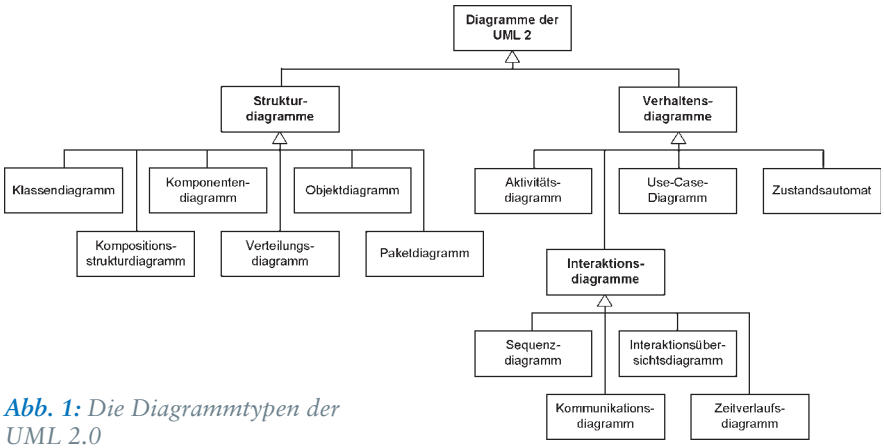

Abb. 1: Die Diagrammtypen der UML 2.0

UML hat sich das Softwareentwicklungsumfeld gravierend verändert. So verdrängten neu aufgekommene Programmiersprachen (hier ist noch Java und noch nicht C# gemeint, dies wird sich vielleicht mit UML 3.0 ändern) die bis dato Vorherrschenden; auch neue Anwendungsparadigmen – wie *Application-Server*-basierte Lösungen – haben inzwischen breite Bedeutung erlangt. Überdies dringen objektorientierte Techniken mit dem Erschließen des Marktes eingebetteter (Realzeit-)Systeme in Anwendungsdomänen vor, die bisher „klassischen" Ansätzen vorbehalten waren.

Ferner wird die Notwendigkeit einer Überarbeitung der UML schon allein durch den an der Modellierungssprache immer stärker zu Tage tretenden „Second System Effect" offenkundig. Dieser klassische Defekt (vgl. [Bro75]) vieler Systeme, die als Verbesserung eines bestehenden Vorgängers konzipiert wurden, macht auch vor der UML nicht Halt, die ursprünglich als erweiterte Vereinigung dreier Methoden konzipiert war (siehe oben). Die UML 1.x versucht viele Details gegenüber ihren Vorgängeransätzen zu verbessern und verliert sich daher leider an vielen Stellen in „Featuritis". Programmiersprachenspezifische Sprachkonstrukte, wie das aus C++ altbekannte friend oder die höchst unterschiedlich interpretierte Semantik der Aggregations- und Kompositionsassoziationen, illustrieren nur zwei der vielen Fälle.

Die Entfernung der programmiersprachen-spezifischen Sprachkonstrukte aus der UML oder zumindest deren erfolgte Verschiebung in optionale, bestimmten Anwendungsfällen vorbehaltene, Sprachanteile markiert gleichzeitig auch den durch die *Model-Driven-Architecture*-Initiative (*MDA*) der OMG verstärkten Trend, die UML als plattformunabhängige, logische Modellierungssprache zu verwenden.

Die Zeiten ändern sich …

Um UML 2.0 nicht in dieselbe Falle vermeintlicher Verbesserungen tappen zu lassen, wurde dem süßen Gift fundamentaler Änderungen widerstanden und stattdessen die Hand (manches Mal auch die Axt) eher an die Wurzel der Sprache gelegt als an ihre, in den verschiedenen Diagrammtypen repräsentierten, sichtbaren Komponenten. Daher wurde die bereits in UML 1.x eingeführte bekannte Notation zur Formulierung statischer Modelle kaum angetastet. So präsentieren sich Klassen-, Objekt- und Verteilungsdiagramm in kaum veränderter Form. Ihre Interna wurden jedoch durch ein nahezu vollständig neu gefasstes Metamodell (d. h. die in UML erfolgte Beschreibung der UML) auf eine stabilere Basis gestellt, die zusätzlich die Bedeutung der einzelnen Modellkonstrukte durch formale Semantikbeschreibungen präzisiert. Für den Sprachanwender offenbart sich dies als doppelter Gewinn. Einerseits konsolidiert das neue Metamodell identische Konzepte durch die Verwendung derselben Metaelemente und fällt daher deutlich kompakter aus als sein schwerfälliger (und sich leider gern widersprüchlich gebender) Vorgänger. Aus diesem Grunde kann die UML 2.0 leichter erlernt und angewendet werden, da die vorhandenen Basiskonzepte in verschiedenen Diagrammen mit demselben Verhalten und identischer Bedeutung eingesetzt werden können. Zum anderen vereinfacht die neue Sprachbeschreibung die Werkzeug- und Methodenentwicklung, die besonders unter bestehenden Widersprüchen und semantischen Unzulänglichkeiten litten.

Mit dem *Kompositionsstrukturdiagramm* wurde ein zusätzlicher Diagrammtyp in die UML aufgenommen (siehe Abb. 1), der besser als bisher die Abbildung vollständiger Architekturen unterstützen soll – daher wird dieses Diagramm häufig auch als *Architekturdiagramm* bezeichnet. Der neue Diagrammtyp zielt auf die Top-Down-Modellierung vollständiger Systeme auf einem geringeren Detaillierungsniveau, als es die implementierungsnahen Konstrukte des Klassendiagramms oder die abstrahierten des Verteilungsdiagramms gestatten.

Die nachhaltigste Änderungen erfährt indes der Teilbereich der dynamischen Modellierung. Er wurde grundlegend entkernt und renoviert, sodass er sich zwar noch der aus den Vorgängerversionen bekannten grafischen Modellelementen bedient, diese jedoch überwiegend mit neuer Bedeutung und Mächtigkeit anbietet.

Die UML 2.0 Verhaltensdiagramme

Das Gros der Standardisierungsarbeit an der UML 2.0 ist in die Verhaltensdiagramme geflossen. Das verwundert die Kenner der UML 1.x mit Sicherheit nicht. Zu groß waren die Mängel in der Semantik, zu eingeschränkt oder unhandlich die Notationsmittel und zu schwach die Anbindung an die statischen Diagramme. Wie auch sonst hätten sich Alternativen von ARIS bis zur SDL behaupten können; sogar das über 30 Jahre alte Nassi-Shneiderman-Struktogramm hatte ohne Wenn und Aber seine Berechtigung. Dem Anspruch, eine universelle Sprache für die meisten Bereiche der Software- und Systementwicklung zu sein, wurde die UML in der Dynamikmodellierung bisher am wenigsten gerecht.

Konzepte statt Diagramme

Nun, mit der UML 2.0 sollte und wird alles anders – versprochen. Wir können Ihnen im Rahmen dieses Artikels zwar nur einen Bruchteil von dem vermitteln, was sich wirklich getan hat, möchten Ihnen aber dennoch die wichtigsten Neuerungen aufzeigen. Dabei kommt uns eine zentrale Idee der UML-2.0-Autoren entgegen: *Die UML 2.0 definiert in erster Linie Konzepte und erst in zweiter Linie Notationselemente und Diagramme.*

Verstehen Sie das nicht falsch, die UML ist und bleibt eine rein grafisch definierte Notation; sie ist weder formal definiert, noch berücksichtigt sie Vorgehensaspekte der Modellerstellung. Die Basiskonzepte legen die Semantik fest, auf welche Diagramme und Notationselemente als Sichten und Filter einwirken. Dieselben Sachverhalte (besser: dieselben Konzepte) werden in verschiedenen Diagramm-

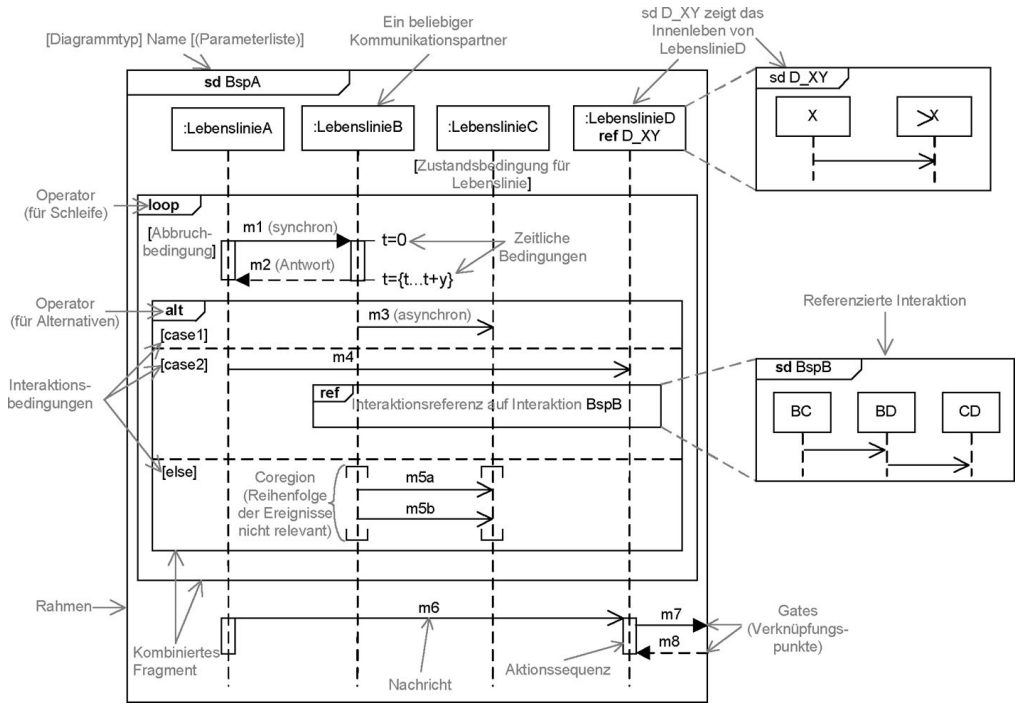

Abb. 2: Die Neuerungen des Sequenzdiagramms

typen unterschiedlich dargestellt oder genutzt. Der Vorteil für Sie ist, dass Sie nur wenige Konzepte erlernen müssen; die konkrete Notation können Sie dann beispielsweise in [Jec03] oder in der einschlägigen Literatur nachlesen. So wissen Sie zum Beispiel nach dem Lesen dieses Artikels, dass die Verhaltensdiagramme schachtelbar sind – und zwar alle sieben (siehe Abb. 1), vom Use-Case- bis hin zum Sequenzdiagramm. Obwohl die Diagramme dabei in ihre Notation variieren, ist das Konzept durchgängig realisiert.

Es ist durchaus möglich und sogar im Standard vorgesehen, dass die UML in den nächsten Jahren noch um weitere Diagrammtypen, sprich Sichten, erweitert wird – es ist jedoch die Zielsetzung die Konzepte dabei stabil zu halten. Natürlich fokussiert ein Diagramm immer einen bestimmten Aspekt und nicht in allen Diagrammen werden alle Konzepte unterstützt, dennoch besticht die UML 2.0 durch ihre Einheitlichkeit und Konformität.

Aus diesem Grund ersparen wir es uns an dieser Stelle, die weit über 100 neuen Notationselemente im dynamischen Bereich im einzelnen vorzustellen, sondern bringen Ihnen die Dynamiknotation der UML 2.0 anhand von Basiskonzepten näher.

Zerlegbarkeit und Verknüpfung von Verhalten

Zunächst ist es sicherlich kein Geheimnis, dass moderne Systeme häufig in ihrer Komplexität und Struktur enorm anwachsen und für einzelne Personen und Teams alleine nicht mehr handhabbar sind. Derartiges überträgt sich auch auf die zugehörigen UML-Modelle, die gerade in der Verhaltensbeschreibung im Detaillierungsgrad sehr schnell anwachsen. Denken Sie an Aktivitäts- oder Sequenzdiagramme, die neben dem Standardablauf einer Ausführung zusätzlich Fehler- und Sonderfälle berücksichtigen können oder Situationen, an denen sehr viele Objekte beteiligt sind. Egal, ob Sie ein System *bottom-up* oder *top-down* modellieren – ab einer gewissen Größe müssen Sie auf verschiedenen Abstraktionsebenen arbeiten. Architekten und Analytiker benötigen einen Grobüberblick, Feindesigner modellieren Details und für einen Codegenerierungsansatz ist in aller Regel ein hoher Präzisionsgrad nötig.

In der UML 2.0 dürfen Sie daher Verhaltensdiagramme beliebig schachteln, d.h. verfeinern und vergröbern. Es ist zudem möglich, gleichartiges, mehrmals beschriebenes Verhalten an nur genau einer Stelle zu definieren und an beliebigen anderen Stellen einzubinden.

Derartige Zerlegungen von Verhaltensbeschreibungen erfordern leistungsfähige Mechanismen zur Konsistenzwahrung. Die hierfür in der UML 2.0 eingeführten Elemente werden im Folgenden dargestellt.

Rahmen und Verhaltensparameter

Um Diagramme oder darin beschriebenes Verhalten referenzieren zu können, muss dies klar umgrenzt und eindeutig identifizierbar sein. Hierzu definiert die UML 2.0 Rahmen (siehe Abb. 2), die Verhaltensbeschreibungen einen Namen geben und die die Angabe von Ein- und Ausgabeparametern (in etwa so, wie Sie es von Methodensignaturen gewohnt sind) ermöglichen. Derart deklariertes Verhalten kann nun an anderer Stelle „aufgerufen" werden. Dabei dürfen Sie beispielsweise aus einem Aktivitätsdiagramm ein mit einem Automaten beschriebenes Verhalten aufrufen. Darüber hinaus sind ab UML 2.0 bei allen Verhaltensaufrufen die Vor- und Nachbedingungen explizit notierbar.

Während die Schachtelung bisher nur bei Use-Cases und Aktivitäten üblich war, wurde sie im Bereich der Zustandsautomaten und Interaktionsmodellierung (Sequenzdiagramme usw.) verfeinert und neu eingeführt. Mittels *Unterzustandsautomatenzuständen* und *Interaktionsreferenzen* lassen sich Automaten bzw. Interaktionen auslagern; zudem dürfen in Sequenzdiagrammen die Lebenslinien zerlegt werden. Dies ist insbesondere dann sinnvoll, wenn die Lebenslinie bei-

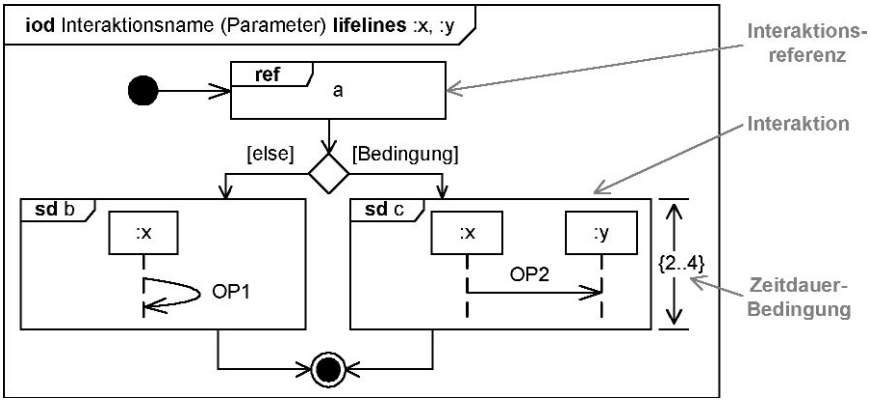

Abb. 3: Eine neue Sicht – das Interaktionsübersichtsdiagramm

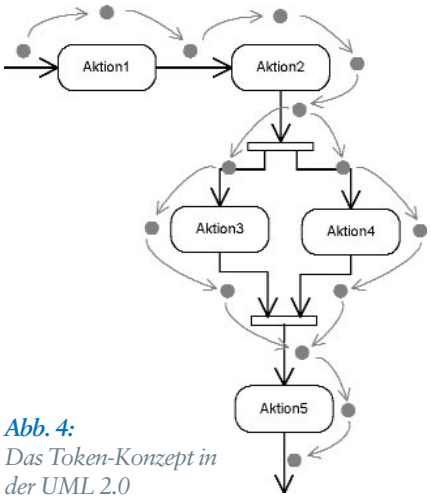

Abb. 4:
Das Token-Konzept in der UML 2.0

spielsweise eine Klasse oder Komponente mit einer komplexen internen Struktur repräsentiert. Zur Darstellung der internen Kommunikationsabläufe (*White-Box*-Ebene) ist ein Zoom von der *Black-Box*-Ebene dann möglich (siehe Abb. 2). Damit dürften ausufernde Sequenzdiagramme der Vergangenheit angehören.

Verknüpfungspunkte

Neben den Verhaltensparametern der Rahmen lassen sich Verhaltenbeschreibungen über spezielle benannte Schnittstellenpunkte miteinander verknüpfen. So dienen *Gates* zur Verbindung von Interaktionsdiagrammen (siehe Abb. 2) und *Ein- und Austrittspunkte* für Unterzustandsautomatenzustände (siehe Abb. 6).

Interaktionsübersichtsdiagramm

Zur Unterstützung eines *Top-Down*-Vorgehens wurde zudem mit dem Interaktionsübersichtsdiagramm (siehe Abb. 3), ein neuer Diagrammtyp eingeführt. Es fädelt unterschiedliche Interaktionsdiagramme (Sequenz-, Timing-, Kommunikationsdiagramme) zusammen und ist von seinem Wesen her ein Aktivitätsdiagramm, das statt Aktionen (Ablaufschritten) Interaktionsdiagramme enthält.

Statt der Ausführung einer Aktion wird ein Interaktionsdiagramm aufgerufen und abgearbeitet. Dadurch haben Sie die Möglichkeit den groben Ablauf in Form einer Aktivität zu modellieren und die einzelnen Schritte als Interaktionen zu verfeinern. Nebenbei verknüpft Ihr Modell die erstellten Interaktionsdiagramme, was in der Vorgängersprachversion nur per Namenskonvention möglich war – ein weiteres neues Element in der UML 2.0, um die Konsistenz im Modell zu sichern.

Ausführbarkeit und semantische Präzision

Um es vorwegzunehmen, auch die UML 2.0 ist in ihrer Vollständigkeit nicht ausführbar, es fehlt nach wie vor eine zu Grunde liegende formale Grammatik. Dennoch wurde hinsichtlich semantischer Präzision einiges getan, wodurch der erste Schritt zu ausführbaren Modellen vollzogen wurde.

Token-Konzept für Aktivitätsdiagramme

Den Aktivitätsdiagrammen liegt mit der UML 2.0 eine aus den Petrinetzen entlehnte *Token*-Semantik (siehe Abb. 4) zugrunde, mit der präzise Regeln für den Ablauffluss, inklusive Parallelisierung, Zusammenführung, *Threading* und Objektfluss geschaffen werden. Damit hat sich das Aktivitätsdiagramm endgültig von den Zustandsautomaten gelöst. (Bei Letzterem hat sich im Übrigen wenig an Semantik und Notation gegenüber der Vorgängerversion verändert. Sie beruhen immer noch auf den *State Charts* nach Harel, vgl. [Har87].) Ein *Token* entspricht genau einem Ablaufpfad, der erzeugt und vernichtet werden kann. Dabei repräsentiert das *Token* entweder das Fortschreiten des Ablauf- oder des Datenflusses. Damit geschaffene Möglichkeiten werden insbesondere in der Geschäftsprozessmodellierung von Nutzen sein (vgl. [Oes03]).

Bedauerlicherweise hat es auch eine Änderung der Begrifflichkeiten gegeben: Ein UML-2.0-Aktivitätsdiagramm (das Zeichenblatt) zeigt nun Aktivitäten (die Verhaltensbeschreibung); eine Aktivität zeigt Aktionen (die Schritte eines Ablaufs). Die aus früheren UML-Varianten bekannten Aktivitäten heißen also jetzt Aktionen und werden überdies noch mit dem gleichen Symbol wie Zustände (Rechteck mit runden Ecken) notiert (siehe Abb. 5).

Ereignismengen in der Interaktionsmodellierung

Im Umfeld der Interaktionsdiagramme hat man die Konzepte der *Message Sequence Charts (MSCs)* (vgl. [ITU99]) übernommen. Diese verfügen unter anderem über ein präzises Ereignismodell, das eine Nachricht mit zwei Ereignissen (einem Sende- und einem Empfangsereignis) in

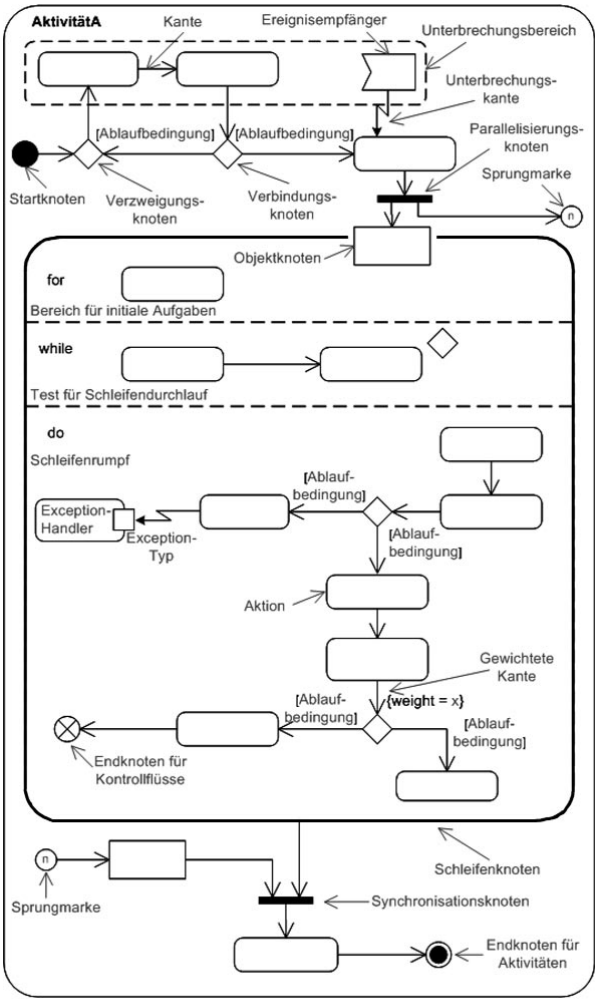

Abb. 5: Einige neue Möglichkeiten in der Aktivitätsmodellierung

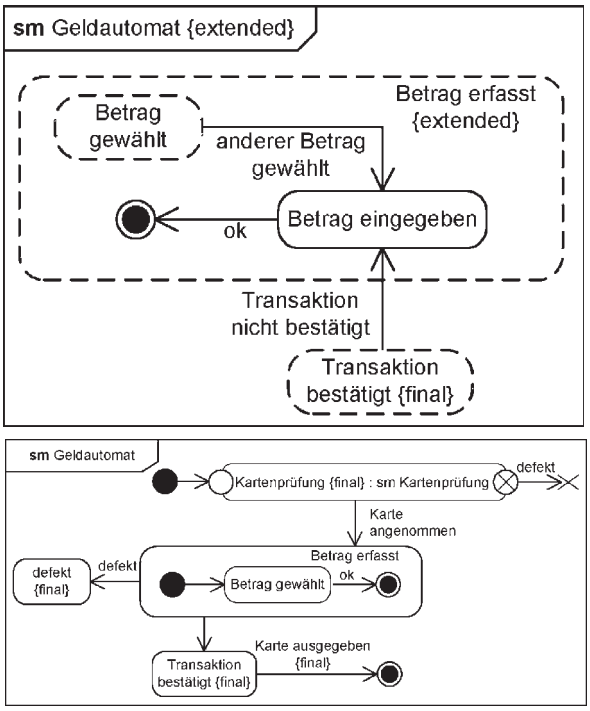

Abb. 6: Die Spezialisierung von Zuständen

Beziehung setzt. Beide Ereignisse sind voneinander entkoppelt, um das Senden und Empfangen getrennt behandeln zu können. Eine Folge von Nachrichten in einer bestimmten zeitlichen Reihenfolge – und nichts anderes zeigen Interaktionsdiagramme – lässt sich somit als geordnete Menge von Ereignissen darstellen. Als Modellierer haben Sie dadurch die Möglichkeit äußerst präzise festzulegen, wann und in welcher Reihenfolge eine Nachricht gesendet bzw. empfangen wird. Nachrichten (Operationsaufruf oder Signalübermittlung) bzw. die zugehörigen Ereignisse lassen sich ordnen, parallelisieren oder in Abhängigkeit setzen (siehe Abb. 2).

Kontrolle der dynamischen Abläufe

Komplexe dynamische Abläufe und Algorithmen laufen in aller Regeln nicht linear ab, sondern werden durch Bedingungen oder Zustände und äußere Faktoren, wie Nutzereingaben oder Berechnungsergebnisse, beeinflusst, d.h. gesteuert, gelenkt, kontrolliert, wiederholt oder abgebrochen. Die UML 2.0 bietet Ihnen dazu alle aus den höheren Programmiersprachen bekannten Primitiven durch eigenständige Notationselemente an. Im Einzelnen sind dies:

- Schleifen (Zähl- sowie kopf- und fußgesteuerte Schleifen),
- Selektionen (Einfach- und Mehrfachselektionen) und
- Ausnahmebehandlungen

Zur Umsetzung dieser Konzepte wurden in Aktivitätsdiagrammen spezielle *strukturierte Knoten* (Schleifenknoten/Entscheidungsknoten) (siehe Abb. 5) und *Exception-Handler* eingeführt. In Sequenzdiagrammen gibt es zum selben Zwecke *kombinierte Fragmente* (siehe Abb. 2), die entsprechende Konzepte in der dort gebräuchlichen Notation abbilden.

Daneben finden sich an vielen Stellen der neuen UML-Fassung kleinere Ergänzungen, mit denen eine bessere Kontrolle und auch Darstellung der dynamischen Abläufe möglich sind. Beispielsweise dürfen Sie *Kanten* in Aktivitätsdiagrammen *gewichten*, um den Ablauf in Abhängigkeit von einer Menge (Anzahl der *Tokens*, Durchläufe, vorliegenden Objekte) zu steuern. *Mengenverarbeitungsbereiche* stellen zudem die sequenzielle, parallelisierte oder iterative Abarbeitung von Mengenstrukturen (Felder, Listen, sonstige geordnete und ungeordnete Containerstrukturen usw.) dar.

Synchronisation mit den Strukturdiagrammen

Die Modellierung von Struktur und Verhalten eines Systems erfordert die Verbindung und Konsistenzwahrung zwischen den statischen und dynamischen Diagrammen. Die UML 2.0 bietet hierfür eine verbesserte Unterstützung: Zum einen, wie bereits erwähnt, durch Zerlegung und Abbildung auf strukturierte *Classifier*, zum anderen durch kleine aber wichtige Details.

Verknüpfung von statischen Elementen und Interaktionen

Durch neue Verknüpfungsmöglichkeiten lassen sich jetzt auf einfache Art und Weise Operationen mit Sequenzdiagrammen beschreiben. Dazu dürfen Sie die Elemente einer Operationssignatur (Parameter und Rückgabewert) als Lebenslinien modellieren. Auch die Notation von lokalen Variablen oder mehrwertigen Attributen (Feldern) als Lebenslinie ist definiert. Eine Abbildung von Teilen (*Parts*), Schnittstellen und *Ports*, sowie eine Selbstreferenz (*self*) ist möglich. Mit den oben erwähnten kombinierten Fragmenten lässt sich damit jeglicher Code-Algorithmus *1:1* abbilden und die Konsistenz zwischen Operationssignatur und Attributen bleibt gewahrt.

Spezialisierung und Generalisierung

Einem *Classifier* (z. B. einer Klasse) dürfen Sie Verhalten in Form eines Zustandsautomaten zuweisen. Da *Classifier* spezialisierbar und generalisierbar sind, muss dies jedoch auf die entsprechende Verhaltensbeschreibung übertragen werden. Dies ist im UML-2.0-Metamodell verankert (siehe hierzu [Bor03]). Sie dürfen alle Verhaltensbeschreibungen (Use-Cases, Aktivitäten, Zustandsautomaten und Interaktionen) spezialisieren und generalisieren, um das Verhalten von *Classifiern* kompakt(er) zu notieren.

Als Beispiel zeigt der obere Teil der Abbildung 6 einen Geldautomaten, dessen Zustand „Betrag erfasst" in einer spezialisierten Form vorliegt (unterer Teil in Abb. 6). Gestrichtelte Zustände beschreiben vererbte Zustände, mit {final} deklarierte Zustände sind nicht redefinierbar. Die Semantik dieser Spezialisierung lässt sich sehr einfach durch den resultierenden Automaten beschreiben, der in Abbildung 7 dargestellt ist.

Mit der Einführung der *Ports* wurde es notwendig, auch deren Verhalten zu beschreiben. Dazu steht Ihnen in der UML 2.0 eine spezielle Variante der Zustandsautomaten – die *Protokollzustandsautomaten* – zur Verfügung. Sie erlauben es Ihnen, die in den *Ports* verwendeten Protokolle, d.h. die korrekte Aufrufreihenfolge von Operationen eines *Ports*, zu beschreiben. Daher unterscheiden sie sich in einigen Details von den reinen Zustandsautomaten. So dürfen Sie an den

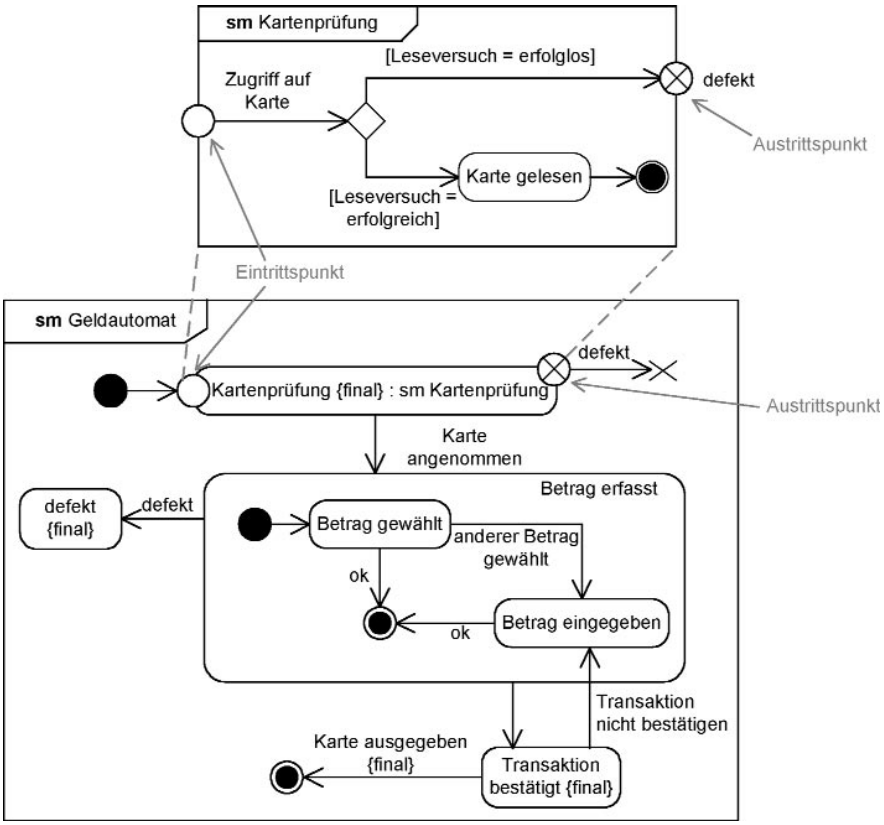

Abb. 7: Der resultierende Automat aus Abb. 6

Transitionen Vor- und Nachbedingungen antragen, die vor bzw. nach der Ausführung der an der Transition angetragenen Operation gelten müssen. Die Zustände dienen als Gedächtnis, um den Fortschritt im Ablauf des Protokolls zu speichern.

Die Zeit als integrales UML-2.0-Konzept

Mit der Aufnahme des Zeitbegriffs als integraler Bestandteil des Sprachumfangs erschließen sich der UML 2.0 völlig neue Anwendungsbereiche. So ist die Betrachtung von Zeitanforderungen gerade für eingebettete und Realzeitsysteme entscheidend. War es in UML 1.x nur möglich, die rein funktionalen Aspekte von Systemen dieses Typs zu beschreiben, so wird die UML durch die in Version 2.0 eingeführten Erweiterungen nun auch bei den Analytikern und Designern solcher Systeme mit Sicherheit eine größere Akzeptanz finden.

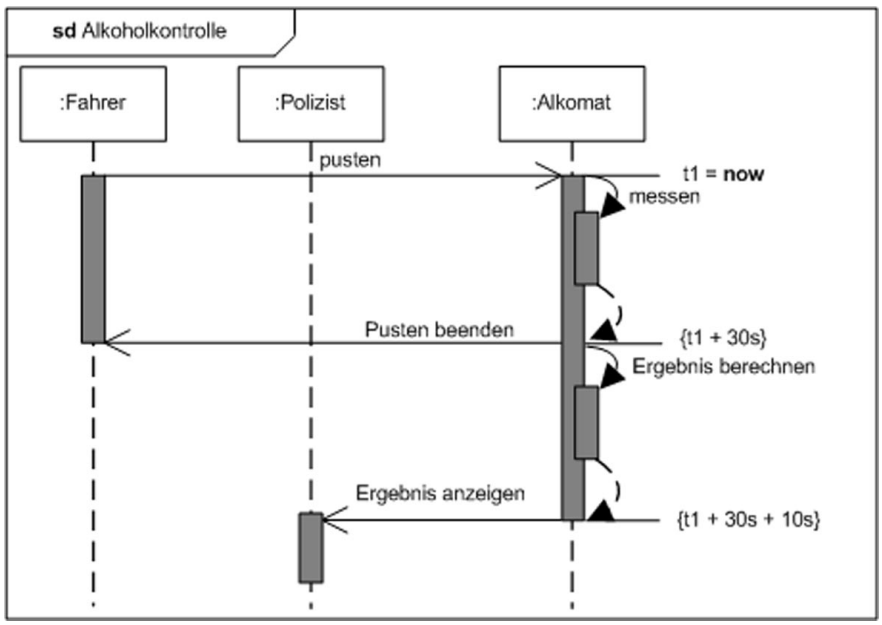

Abb. 8: 30 Sekunden lang pusten

Im Detail bedeutet die Berücksichtigung des Faktors „Zeit", dass es Ihnen nun möglich ist, eine Aktion oder ein Ereignis mit einer Zeit oder einer Zeitdauer zu notieren. Während in den UML-1.x-Versionen diese Annotationen noch recht stiefmütterlich neben den definierten Notationselementen standen, sind nun – nicht zuletzt durch das Zusammenwachsen der einzelnen Diagrammtypen – die Zusammenhänge präziser definiert.

Als Beispiel haben wir in Abbildung 8 in einem Sequenzdiagramm modelliert, wie eine Alkoholkontrolle ablaufen kann (ohne auf den Promillegehalt näher einzugehen). Zum Zeitpunkt `t1` beginnt die Aktion `messen`. Dieser Zeitpunkt wird als Referenzzeitpunkt für das weitere Diagramm herangezogen und mit dem Schlüsselwort `now` bezeichnet. Nach 30 Sekunden ist die Messung beendet und die Berechnung des Ergebnisses kann beginnen. Nach weiteren 10 Sekunden steht das Ergebnis fest und wird dem Polizisten mitgeteilt.

Timing-Diagramm

In einem Timing-Diagramm steht das zeitliche Verhalten der beteiligten *Classifier* im Vordergrund. Das bedeutet, Sie können genau darstellen, wann ein *Classifier* z.B. seinen Zustand ändern soll oder wie lange er für eine Aktion benötigt.

Bei der Definition des Timing-Diagramms wurden einige Anleihen aus der Elektrotechnik vorgenommen, wo diese Art von Beschreibungen schon lange erfolgreich eingesetzt werden. Das Diagramm stellt einzelne *Classifier* (in Abb. 9 der Fahrer, der Polizist und der Alkomat), ihre verschiedenen Zustände und die Kommunikation zwischen ihnen dar. Die Zeit ist auf der horizontalen Achse aufgetragen. Somit ergibt sich ein zeitlicher Ablauf von links nach rechts in dem Diagramm.

Verbesserter Umgang mit Diagrammen

Neben den verbesserten inhaltlichen Konzepten wurde auch einiges für den Zeichenalltag getan. *Sprungmarken* vermeiden längere Kanten in Aktivitätsdiagrammen und ersparen aufwändige Layout-Tätigkeiten. *Unterbrechungsbereiche* (ebenfalls in Aktivitätsdiagrammen) gruppieren Aktionen und ermöglichen den Sprung aus einer beliebigen gruppierten Aktion, auch wenn an dieser keine passende Ausgangskante modelliert wird. Es genügt den Unterbrechungsbereich mit einer Kante zu versehen; diese erstreckt sich analog ▶

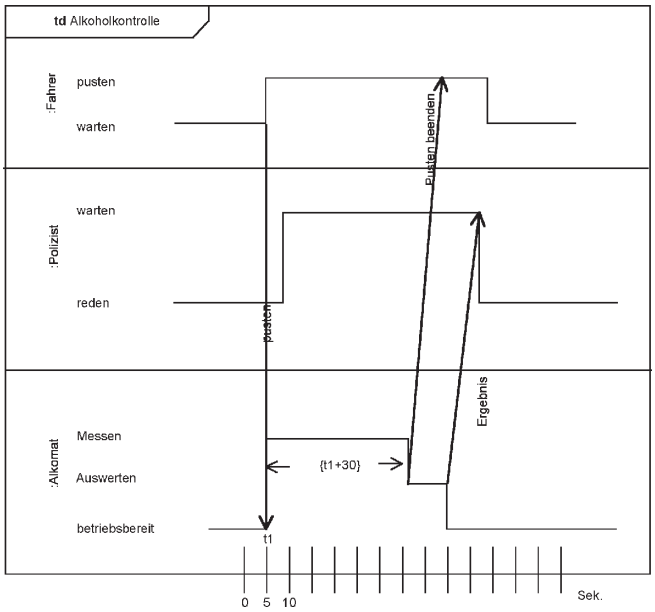

Abb. 9: Die Alkoholkontrolle aus Abb. 8 als Timing-Diagramm

auch auf alle gruppierten Aktionen. Dies ist besonders effektiv für asynchrone Ereignisse, beispielsweise für die Darstellung von Fehlerfällen, die in *jeder* Aktion der Aktivität auftreten können. Anstatt mehrfach von jeder Aktion eine ausgehende Kante anzutragen, gruppieren Sie die entsprechenden Aktionen und modellieren nur eine einzige Kante.

Und wo hat sich nichts getan?

Obwohl die Version 2.0 einen großen Revisionsschritt der Sprache darstellt und insbesondere die Verhaltensdiagramme generalüberholt wurden, existieren auch Bereiche, in denen sich nichts oder für den Praktiker kaum Spürbares verändert hat, beispielsweise bei den *Use-Case-Diagrammen*. Wie bisher besteht die Möglichkeit, die wichtigen Use-Cases Ihres Systems mit den Akteuren in Zusammenhang zu setzen. Der Begriff des „Systems" wurde dabei deutlicher gefasst. Faktisch kann jeder *Classifier* (also auch Klassen, Komponenten oder Geräte) Use-Cases realisieren.

Auch die *Kommunikationsdiagramme* sind im eigentlichen Sinne, abgesehen von ihrem Namen, nicht neu. Bisher hießen sie Kollaborationsdiagramme, obwohl sie eigentlich schon immer Interaktionen zeigten und nicht nur UML-Kollaborationen. Anzumerken ist, dass sie nicht mehr, wie in der Vorgängersprachversion, vollständig äquivalent zum Sequenzdiagramm sind, sondern nur eine Teilmenge der dort darstellbaren Konzepte fokussieren.

Fazit und Resümee

Wie Sie sehen, hat sich einiges getan in der UML. Es gibt kaum ein Diagramm, an dem sich durch den Versionssprung auf 2.0 nichts geändert hat. Dabei fallen folgende Bereiche besonders auf:

- Die Infrastruktur der UML wurde grundlegend überarbeitet. Dadurch ist die Modellierungssprache kompakter und in sich schlüssiger geworden, was der Wiederverwendbarkeit zugute kommt.
- Die Definition der Diagrammtypen enthält zum jetzigen Zeitpunkt noch einige Inkonsistenzen und Widersprüche.
- Einige Notationselemente wurden in ihrer grafischen Repräsentation überarbeitet und eingeführte Bezeichnungen geändert, leider nicht immer zugunsten der Benutzerfreundlichkeit (so werden beispielsweise Aktionen – ehemals Aktivitäten – durch dasselbe Symbol wie Zustände dargestellt).
- Diagramme auf Basis des alten 1.x-Standards lassen sich nicht in jedem Fall verlustfrei übernehmen, da die angestrebte Abwärtskompatibilität zu älteren UML-Versionen nicht in allen Bereichen erreicht wurde.

Damit drängt sich natürlich die Frage auf, für wen es sich lohnt, auf den neuen Standard umzusteigen. Diese Frage lässt sich naturgemäß nicht pauschal beantworten. Aber es gibt Indikatoren, die Ihnen die Entscheidung leichter machen. Sie sollten einen Umstieg auf den neuen Standard dann ernsthaft in Erwägung ziehen, wenn mindestens einer der folgenden Punkte auf Ihr Projekt oder Unternehmen zutrifft:

- Sie modellieren häufig dynamische Aspekte des Systems und verwenden daher oft Zustandsautomaten und Sequenzdiagramme.
- Sie legen Wert auf die Darstellung der Kommunikation von Systemkomponenten (z. B. über *Ports*) oder des Zeitverhaltens Ihres Systems.
- Sie arbeiten viel mit Aktivitätsdiagrammen, z. B. bei der Modellierung von Geschäftsprozessen oder Aktivitäten des Systems.
- Sie arbeiten in der Codegenerierung mit einem MDA-Ansatz oder in einem sich schnell ändernden, architektonischen Umfeld wie *Enterprise Application Integration (EAI)*.

Bevor Sie umsteigen, sollten Sie allerdings die UML-2.0-fähige Version Ihres Modellierungswerkzeuges auf Herz und Nieren testen, denn Tool-Probleme können die Vorteile der neuen Sprachversion schnell zunichte machen. In weiser Voraussicht kommt der neue Standard den Tool-Herstellern hier entgegen: Die UML ist unterteilt in drei verschiedene Erfüllungsebenen (*Compliance Levels*) und diese wiederum in Erfüllungspunkte (*Compliance Points*). Dahinter steckt die Idee, dass auch weniger UML immer noch UML ist; ein Tool-Hersteller muss nicht den gesamten Standard umsetzen, sondern hat die Möglichkeit der schrittweisen Implementierung der UML 2.0. Inwieweit ein Werkzeug die UML 2.0 bereits unterstützt, lässt sich neben den Erfüllungspunkten an vier verschiedenen Erfüllungsgraden (nicht erfüllt, teilweise erfüllt, vollständig erfüllt, Austauschformat erfüllt) ablesen, die ein Hersteller angeben muss. Informationen über die UML 2.0-Fähigkeit von Tools finden Sie in Kürze unter www.uml-glasklar.de. Wenn Sie (noch) kein ausgereiftes UML 2.0-Werkzeug, sondern einfach nur schnelle Hilfe beim Erstellen von Diagrammen benötigen, finden Sie unter www.sophist.de eine Schablone für das Zeichenprogramm „MS Visio", die alle wesentlichen grafischen Neuerungen der UML 2.0 enthält.

Des Weiteren sollten Sie Ihr Team durch einen UML 2.0-erfahrenen Coach in einem Update-Training auf den aktuellen Stand bringen lassen, da sich in einigen Bereichen nicht nur die Notation geändert hat, sondern auch methodisch neue Konzepte zur UML hinzugekommen sind. Erstmals haben Sie auch die Möglichkeit, Ihre UML-Kenntnisse in weltweit einheitlichen Tests nachzuweisen, denn neben dem neuen Standard hat die

OMG ein Zertifizierungsprogramm in drei Stufen erarbeitet (OMG-Certified UML Professional Fundamental/Intermediate/Advanced).

Im Großen und Ganzen kann man die Überarbeitung der UML als gelungen bezeichnen. Von den Änderungen profitieren spezielle Anwendungsbereiche, wie die Entwicklung von eingebetteten oder Realzeitsystemen, aber auch der „normale" UML-Anwender profitiert von neuen Möglichkeiten, beispielsweise einer besseren Verständlichkeit durch die Überarbeitung der Hintergrundkonzepte oder durch sich eröffnende Austauschmöglichkeiten zwischen den verschiedensten Werkzeugen. ■

Literatur & Links

[Bor03] M. Born, E. Holz, O. Kath, Softwareentwicklung mit UML 2, Addison-Wesley 2003

[Bro75] F. P. Brooks, Vom Mythos des Mann-Monats, MITP 2003

[Har87] D. Harel, Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming (8), Elsevier, 1987

[Jeckle] Homepage von Mario Jeckle: www.jeckle.de

[ITU99] International Telecommunication Union, Formal Description Techniques – Message Sequence Charts, ITU-T Recommendation Z.120. 11/1999

[Jec03] M. Jeckle, C. Rupp, J. Hahn, B. Zengler, S. Queins, UML 2 glasklar, Hanser Verlag 2003 (siehe auch: www.uml-glasklar.de)

[Oes03] B. Oestereich, C. Weiss, C. Schröder, T. Weilkiens, A. Lenhard, Objektorientierte Geschäftsprozessmodellierung mit der UML, dpunkt.verlag 2003

[SOPHIST] Homepage der Firma Sophist: www.sophist.de