Zeitschrift: bulletin.ch / Electrosuisse
Herausgeber: Electrosuisse
Band: 95 (2004)
Heft: 1

Artikel: Unified Modeling Language
Autor: Jeckle, Mario / Zengler, Barbara
DOI: https://doi.org/10.5169/seals-857895

Unified Modeling Language

Die neue Version der Modellierungssprache wartet mit verschiedenen Neuerungen auf

Die Unified Modeling Language (UML) ist in den letzten Jahren zur Standardnotation für Analyse und Entwurf von Softwaresystemen avanciert. Während in den letzten Versionen der Modellierungssprache keine grossen Änderungen zu verzeichnen waren, weist die für das Frühjahr 2004 erwartete Revision zur Version 2.0 umfangreiche Neuerungen auf. So werden neben der Aufnahme längst überfälliger Notationselemente in die Modellierungssprache auch gänzlich neue Diagramme eingeführt. Inwieweit hat sich nun die UML geändert? Sollten bestehende Projekte nun auf UML 2 umgestellt werden – und lohnt sich dieser Umstieg? Und wie sieht es mit neuen Projekten aus? Diese und ähnliche Fragen versucht dieser Artikel zu beantworten und damit Hilfestellung für die Praxis zu geben.

Mario Jeckle, Barbara Zengler

Das Einsatzgebiet der Unified Modeling Language (UML) erstreckt sich von der Modellierung und Spezifizierung von komplexen Systemen – wobei es sich dabei nicht notwendigerweise um Software handeln muss – bis hin zu deren Dokumentation und Visualisierung. Ihre Standardisierung erfolgt durch die Object Management Group[1] (OMG), einem Industriegremium mit etwa 800 Mitgliedern, unter denen sich namhafte Firmen wie Borland, DaimlerChrysler, Gentleware, Oracle, Rational Software (IBM), SAP, Telelogic u.a. befinden. Diese breite Unterstützung seitens der Hersteller und späteren Anwender ist unabdingbare Basis für die breite Akzeptanz dieses herstellerneutralen Standards.

Geschichte der UML

Die UML entstand als direkte Folge der «Methodenkriege der Softwareentwicklung» in den frühen 90er-Jahren. Zu dieser Zeit konkurrierte eine Vielzahl objektorientierter Methoden und Modellierungsmöglichkeiten offen am Markt. Unter den «berühmtesten» – gemessen an Akzeptanz und Verbreitungsgrad – auch Rumbaughs OMT[2], Jacobsons OOSE[3], Boochs OOD[4] u.a. (Bild 1). Als Konsequenz der herrschenden Vielfalt erschwerten die unterschiedlichen grafischen Darstellungsformen der jeweiligen Methoden die Kommunikation zwischen Systementwicklern entscheidend und trugen letztlich zur mangelnden Breitenakzeptanz der Ansätze bei[5].

Genau genommen mussten die Systementwickler zunächst die Eignung der jeweiligen Methode für jedes Einzelprojekt bewerten. Zusätzlich standen sie vor der Frage, welche der Techniken in zwei Jahren noch Bedeutung am Markt haben würde. Aus dieser Unsicherheit heraus verzichteten viele Unternehmen völlig auf den Umstieg zu objektorientierten Modellierungsansätzen und setzten weiterhin traditionelle Analyse- und Entwurfsmethoden ein.

In den Jahren 1997 und 1998 wurden einige der verschiedenen Notationen zur UML in der Version 1.0 vereinigt – vor allem die Ansätze von Rumbaugh, Booch und Jacobson, die sich zuvor schon durch eine starke Konfluenz der verwenden Konzepte auszeichneten. Grosse Unterstützung erfuhr die UML zu dieser Zeit durch die Firma Rational, einem grossen Hersteller von Softwareentwicklungswerkzeugen. Dieser bezahlte die drei Initiatoren der UML[5] für ihre Arbeit an der Modellierungssprache und förderte so die schnelle Entstehung des Standards.

In den folgenden Jahren durchlief die UML dann weitere Reifungsstadien, etwa durch Integration der Object Constraint Language (in Version 1.1) zur formalisierten Angabe von Konsistenzbedingungen und die Hinzunahme des Modellaustauschformates XML Metadata Interchange[6] (in Version 1.3).

Zur selben Zeit übernahm die OMG das Copyright an der UML. Durch diesen entscheidenden Schritt gewann die Notation ihre Unabhängigkeit von einzelnen Unternehmen und konnte sich in der Folge als Standard innerhalb des Softwareentwicklungsmarktes etablieren.

Die Folgeversionen UML 1.4 und 1.5 fanden in vielen Bereichen Einsatz und erlebten weitere Veränderungen, vorangetrieben durch die aus der Anwendung gewonnenen Erfahrungswerte. Durch diese «Weiterentwicklungen» wurde die UML jedoch sehr aufgebläht und bietet inzwischen einige Ansatzpunkte zur Kritik.

Warum eine neue UML-Version? – Anforderungen an die UML 2

Als bedeutendster Kritikpunkt der UML 1.x sind deren Umfang und Komplexität zu nennen, die ein schnelles Erlernen und eine angemessene Benutzung der Sprache erschweren. Darüber hinaus stellen Veränderungen des Softwareentwicklungsmarktes neue Anforderungen an eine zeitgemässe Modellierungssprache – so wird diese in neuen Anwendungsbereichen eingesetzt, etwa der Modellierung von Systemen mit Realzeitanforderungen. Hinsichtlich dieser gewandelten Bedingungen ist eine Verbesserung der UML wünschenswert.

Skeptiker befürchten jedoch umfangreichere Möglichkeiten auf Kosten einer noch höheren, unerwünschten Komplexität. Dieses Problem ist seit mehr als 30 Jahren unter dem Schlagwort vom «Second System Syndrom» bekannt: Zweitsysteme entwickeln sich durch Hinzunahme vieler Ideen und Neuerungen meist zu einem riesigen undurchschauba-

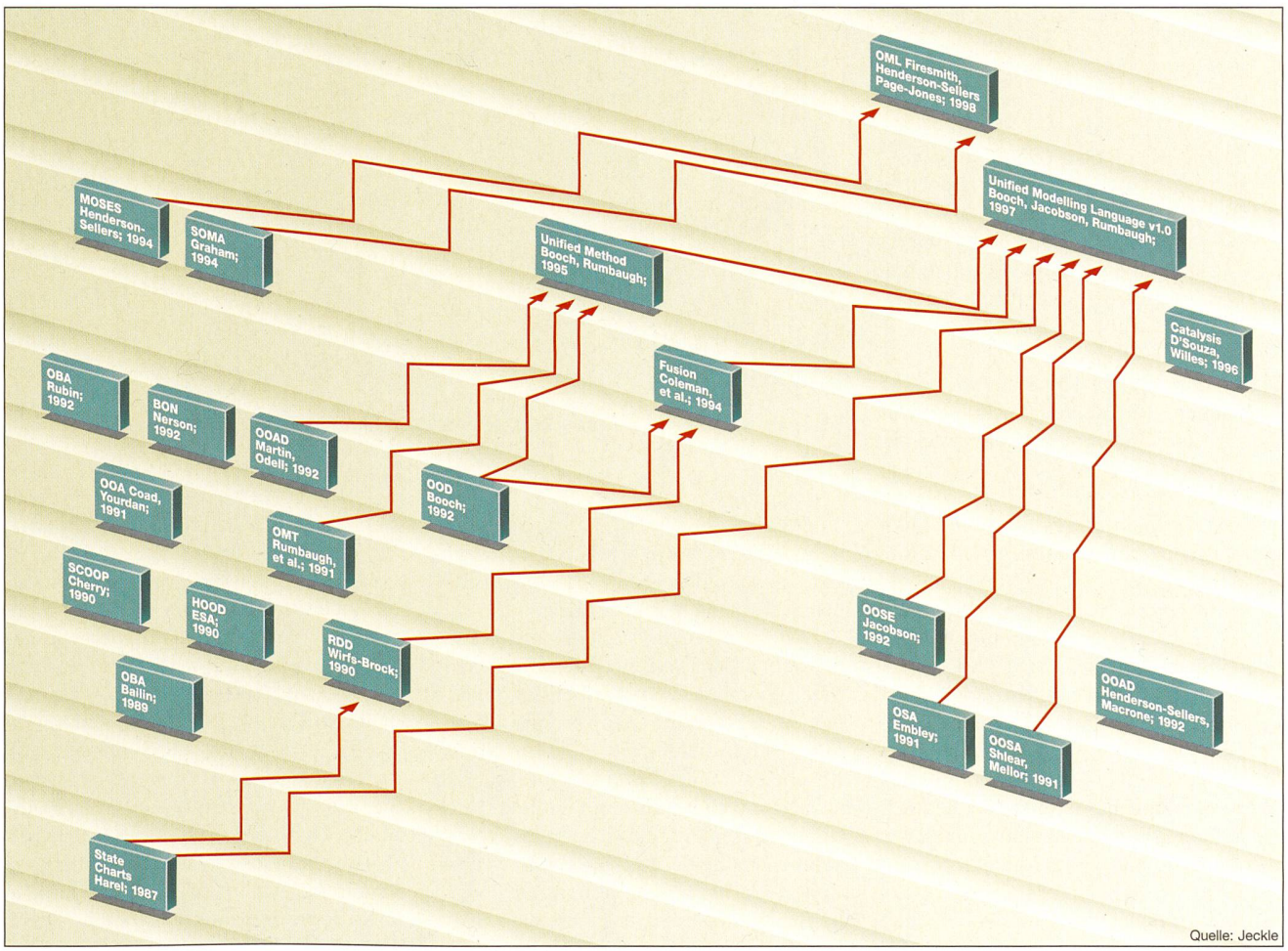

Bild 1 Auf der Suche nach der «Lösung» – der Methodenkrieg

BON: Better Object Notation; HOOD: Hierarchical Object Oriented Design; MOSES: Methodology for Object Oriented Software Engineering of Systems; OBA: Object Behaviour Analysis; OML: Open Modeling Language; OMT: Object Modeling Technique; OOA: Object Oriented Analysis; OOAD: Object Oriented Analysis and Design; OOD: Object Oriented Design; OOSA: Object Oriented System Analysis; OOSE: Object Oriented Software Engineering; OSA: Object Oriented System Analysis; RDD: Responsibility Driven Design; SCOOP: Software Construction by Object-Oriented Pictures; SOMA: Semantic Object Modelling Approach.

ren Gebilde. Dieses Phänomen sollte für die zweite Version der UML unbedingt vermieden werden.

Nach vielen Diskussionen innerhalb der OMG kristallisierten sich die folgenden Forderungen als zentrale Punkte für die Weiterentwicklung der UML heraus. [1]

Die UML 2 soll:

- die komponentenbasierte Entwicklung (zum Beispiel mit J2EE[7]) besser unterstützen;
- die Spezifikation von Echtzeitarchitekturen durch das Hinzufügen geeigneter Notationsmittel ermöglichen, um damit den Einsatz der UML in technischen Systemen zu erlauben;
- die in Teilbereichen noch schwach definierte Semantik (z.B. die der Aggregation und Komposition) klarstellen;
- Kapselung und Skalierbarkeit in der Verhaltensmodellierung – ganz besonders bei Zustandsautomaten und Interaktionsdiagrammen – besser unterstützen;
- die bestehenden Einschränkungen bei der Aktivitätsmodellierung aufheben.

Als «Streich»-kandidaten, d.h. Elemente, die im Zuge der Erneuerung aus dem Sprachumfang entfernt werden, boten sich vor allem drei Arten von Sprachbestandteilen an [1], nämlich UML-Elemente,

- die weder von den wichtigsten Werkzeugherstellern implementiert werden, noch in etablierten Vorgehensmodellen Verwendung fanden;
- die entweder methoden- oder implementierungsspezifisch sind (z.B. spezifische Sprachkonstrukte wie der C++ friend-Stereotyp);
- denen eine präzise Semantik fehlt (z.B. copy- und become-Stereotypen, refine- und trace-Stereotypen).

Der ausgereifteste Änderungsvorschlag wurde durch die «U2 Partners», ein Konsortium bestehend aus einer Reihe von Werkzeugherstellern, wie Computer Associates, Hewlet-Packard, IONA, Oracle, Rational Software, Unisys, IBM, Telelogic, verstärkt durch Unternehmen, welche die UML im grossen Stile einsetzen (darunter Ericsson, France Telecom, DaimlerChrysler), erarbeitet.

Die wichtigsten Ziele bei dieser Überarbeitung der UML sind *Präzisionssteigerung*, *Ausführbarkeit* und vor allem *Übersichtlichkeit*:

- Eine präzisere UML erfordert eine Neuformulierung des Metamodells, sowie eine weitest gehende Verwendung der Object Constraint Language (OCL) und – soweit sinnvoll – eine möglichst unveränderte Wiederverwendung von Basiskonstrukten.
- Die Verbesserung der Ausführbarkeit setzt eine Erweiterung der Zustandsautomaten und vor allem eine stärkere

Beziehung zwischen statischen und dynamischen Diagrammen voraus. Weitere Verbesserungsansätze bieten sich durch Integration erprobter Konzepte (wie beispielsweise Petri-Netze).
- Eine übersichtlichere UML fördert als wichtigstes Ziel die Verständlichkeit und verbessert somit die Kommunikation zwischen Projektteilnehmern. Möglich ist dies durch Verringerung der verwendeten grafischen Modellkonstrukte und Basiskonzepte.

Aus diesen Ideen heraus entstanden im Wesentlichen vier sich ergänzende und ineinander verzahnte UML-2-Dokumente: *Infrastructure*, *Superstructure*, *Object Constraint Language* (OCL) und *Diagram Interchange*.

Das *Infrastructure*-Dokument hält grundlegende Sprachkonstrukte und die Basisarchitektur fest, wohingegen das *Superstructure*-Dokument aufbauend auf dieser Architektur Diagrammnotation und Semantik enthält. Das *OCL*-Dokument beschreibt eine auf dem Prädikatenkalkül fussende formale Sprache zur Formulierung konsistenzgarantierender Einschränkungen. Das *Diagram Interchange*-Dokument definiert schliesslich die Metamodellelemente und Grundlagen zum Austausch visueller Modellrepräsentationen. Die Dokumente – ausser OCL – befinden sich zurzeit in der Annahmephase (die Bestätigung durch das Board of Directors steht noch aus). Daher sind in diesen Dokumenten keine grossen Änderungen mehr zu erwarten – lediglich in bisher vernachlässigten oder fehlerhaften Teilbereichen sind Nachbesserungen vonnöten.

Bedingt durch die grosse Komplexität der UML und die notwendige Sorgfalt zur Auswahl der Änderungen und Anpassungen, ist die ursprüngliche Zeitplanung um etwa 11/2 Jahre nach hinten verschoben worden (Bild 2, [2]).

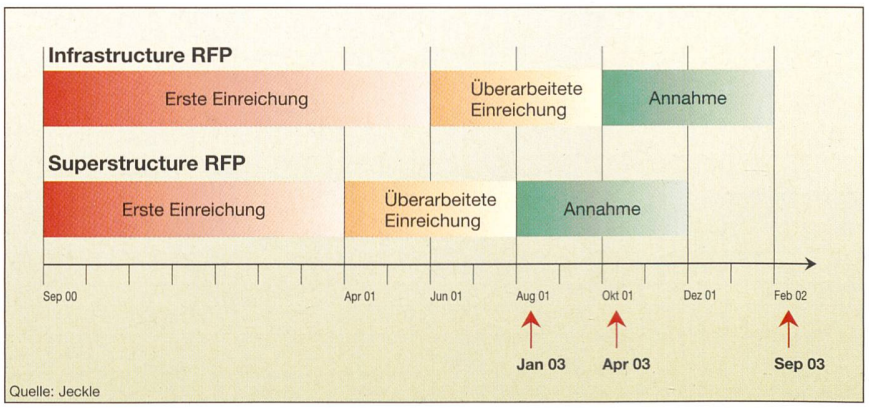

Bild 2 Der UML-2-Ablaufplan – purer Zweckoptimismus, eine Konstante im Software Engineering

UML-Erfüllungsgrade

Der modulare Aufbau der so genannten UML-Erfüllungsgrade (engl. *compliance levels*) erlaubt es den Herstellern von UML-Werkzeugen, ihre Produkte speziell auf den gewünschten Zielmarkt auszurichten.

Eine grobe Einteilung der Modellierungskonstrukte in die drei Erfüllungsgrade *basic* (Level 1), *intermediate* (Level 2) und *complete* (Level 3) gruppiert die einzelnen Bestandteile der UML, welche die konzeptuellen Konstrukte der Sprache widerspiegeln.

Es ist also nicht nötig, alle Modellelemente der UML in ein Werkzeug zu integrieren, um dieses spezifikationskonform zu gestalten. Hierfür ist lediglich die Implementierung des UML-Kerns (engl. *kernel*), der einen Teil des Erfüllungsgrades *basic* darstellt, zwingend nötig.

Den Werkzeugherstellern wird damit ermöglicht, die von ihren Kunden benötigten Komponenten flexibel auszuwählen (z.B. Interaktionen, Anwendungsfälle, Strukturelemente) und in ihrem Kernbereich einen hohen Erfüllungsgrad anzubieten. Beispielsweise wird ein Hersteller, dessen Produkt sich auf die Unternehmensmodellierung fokussiert, erweiterte Aktivitätsdiagramme zur Modellierung von Geschäftsabläufen extensiv einsetzen und daher den höchsten Erfüllungsgrad umsetzen. Dagegen bieten die Zustandsautomaten zur Modellierung interner Zustände kaum eine Hilfestellung für die Abbildung von Geschäftsabläufen, weshalb deren basale Unterstützung in Form des Erfüllungsgrades *basic* genügt. Im Gegensatz dazu wird ein Hersteller von Werkzeugen für Echtzeitsysteme auf erweiterte Zustandsautomaten und nur auf grundlegende Aktivitätsdiagramme zurückgreifen (Bild 3, [1]).

Je nach Projekteinsatz können die UML-Elemente anhand der drei Erfüllungsgrade in unterschiedlich komplexer Darstellung verwendet werden, wodurch sich ein flexibler UML-Begriff am Markt etablieren kann, der zeigt, dass auch die Unterstützung einer geringen Konstruktmenge hinreichend sein kann.

Die Diagrammtypen der UML 2

Die Diagramme der UML 2 wurden – teils im Detail, teils komplett – überarbeitet (Bild 4):

Als mögliche Alternative oder Ergänzung zur grafischen Modellierung von *Verhaltensdiagrammen* steht nun eine tabellenartige Darstellung des Verhaltens zur Verfügung. Diese tabellarische Form kann etwa als Basis für die Generierung von Testfällen hilfreich sein.

Aktivitätsdiagramme haben die meisten Änderungen erfahren. Die wohl wichtigste ist, dass sie keine Sonderform des Zustandsdiagramms mehr sind, sondern auf erweiterten Petri-Netzen basieren. Damit wurden sie von vielen Einschränkungen befreit, was sich vor allem in einer verbesserten Testbarkeit und durch erkennbare Verklemmungsfreiheit ausdrückt. Darüber hinaus werden parallele Flüsse besser unterstützt und Aktivitätsdiagramme sind (fast ohne Hinzufügen zusätzlicher Semantik) ausführbar. Durch Wegfallen alter Restriktionen, Einführung neuer Elemente und Verwendung von Tokens als Basiskonzept des Ablaufs, hat der Modellierer nun eine weitaus grössere Flexibilität bei der Flussmodellierung.

Das *Kommunikationsdiagramm* (das ehemalige Kollaborationsdiagramm) stellt Interaktionen im Lebenszyklus einzelner Komponenten in den Vordergrund, ist also eine Variante des Sequenzdiagramms, wobei der kooperative und weniger der zeitliche Aspekt des Nachrichtenaustauschs veranschaulicht wird. Hier gibt es keine wesentlichen Änderungen zu den UML-Vorgängerversionen.

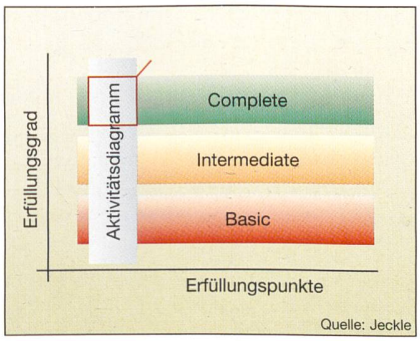

Bild 3 Wie viel wird aus welchen Diagrammen für das Projekt benötigt?

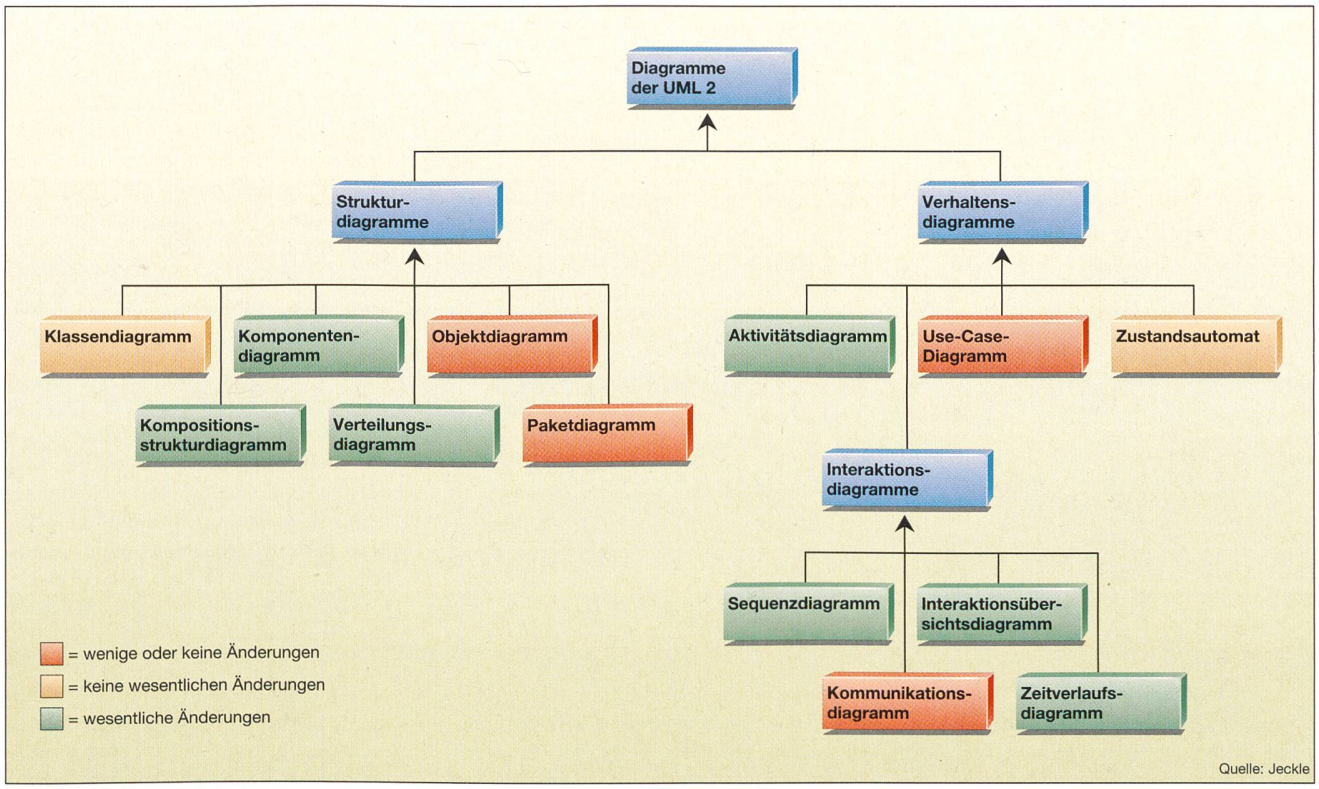

Bild 4 Die Diagrammtypen der UML 2

Bei den *Klassendiagrammen* wurden einige Unsauberkeiten und unpräzise Elemente (z.B: `copy-` und `become-`Stereotypen) neu gefasst und überarbeitet. Die Semantik bei einigen Details hat sich ebenfalls verändert (z.B. geordnete Reihenfolge von Attributen und Operationen in Klassen). Ansonsten bleibt alles beim Alten.

Das *Sequenzdiagramm* ist nun strukturier- und zerlegbar. Es gibt zudem viele Möglichkeiten zur Steuerung der Kontrollflüsse und Nebenläufigkeiten (wie Verzweigungen und Schleifen). Damit lassen sich nicht nur einzelne willkürlich herausgegriffene Szenarien modellieren, sondern auch ganze Verhaltensbereiche. Die neuen Sequenzdiagramme übernehmen alle wesentlichen Konstrukte aus den im technischen Bereich so beliebten Message Sequence Charts[8] (MSC).

Das neu gefasste *Verteilungsdiagramm* zeigt die Laufzeitaspekte einer Architekturumsetzung. Hierzu zählen besonders die Kommunikationsbeziehungen zwischen den Einzelkomponenten. Änderungen betreffen vor allem eine bessere Unterscheidung zwischen Soft- und Hardwareausführungsumgebungen und die Verwendung von Artefakten (z.B. für Spezifikation von Dateien bei der Codegenerierung).

Das neu eingeführte *Kompositionsstrukturdiagramm* zeigt sowohl die Interna von Systemteilen als auch deren Zusammenspiel in Form von zur Laufzeit verarbeiteten Objekten, die über entsprechende Kommunikationswege zusammenarbeiten, um ein gemeinsames Ziel zu erreichen. Das gemeinsame Ziel kann hierbei die Realisierung eines Use Case sein, aber auch die Umsetzung eines Sequenzdiagrammes.

Durch die neu eingeführten *Zeitverlaufsdiagramme* (Timingdiagramme) wird eine präzise Beschreibung des Zeitverhaltens von Objekten und Systemen ermöglicht.

Das Hauptaugenmerk des neu eingeführten *Interaktionsübersichtsdiagramms* liegt auf der Abfolge mehrerer Verhaltensmodelle. Dies ermöglicht die Verwaltung und Integration von Modellen und Diagrammen auf einer hohen Abstraktionsebene.

Das *Komponentendiagramm* wurde ebenfalls neu gefasst und zeigt die Laufzeitaspekte einer Architekturumsetzung. Hierzu zählen vor allem die Kommunikationsbeziehungen zwischen den Einzelkomponenten. In der UML 2 wird die Modellierung gängiger Marktstandards (.NET und J2EE) besser unterstützt.

Zustandsautomaten ermöglichen die zustandsbasierte Verhaltensmodellierung. Die Zustandsmodelle der UML 2 erlauben eine verbesserte Verknüpfung von statischen Elementen (Objekte, Schnittstellen, Komponenten) und dahinter liegenden Zustandsmodellen. Zur präzisen Definition von Schnittstellen wurden Protokollzustandsautomaten eingeführt.

Die weiteren Diagramme (*Use-Case-*, *Objekt-* und *Paketdiagramm*) wurden nur unwesentlich verändert.

Anforderungen eingehalten? Die Bewertung

Obwohl die letztgültige Freigabe des neuen UML-Standards noch aussteht, ist bereits deutlich erkennbar, in welche Richtung die Reise geht und was den Anwender erwartet. Details, die noch in der letzten Annahmephase verändert werden, werden das Gesamturteil kaum beeinträchtigen.

- Die UML ist in der neuesten Version kompakter und in sich schlüssiger geworden. Durch Anpassen von grundlegenden Konzepten der Infrastruktur ist ein wichtiger Schritt in Richtung Wiederverwendung und Straffung der Konzepte getan.
- Die angestrebte Abwärtskompatibilität zu älteren UML-Versionen wurde in einer Reihe von Bereichen verletzt. Daher besteht der begründete Verdacht, dass mit UML-1.x-Werkzeugen erstellte Modelle nicht verlustfrei in die neue Werkzeuggeneration übernehmbar sein werden. Sollten die

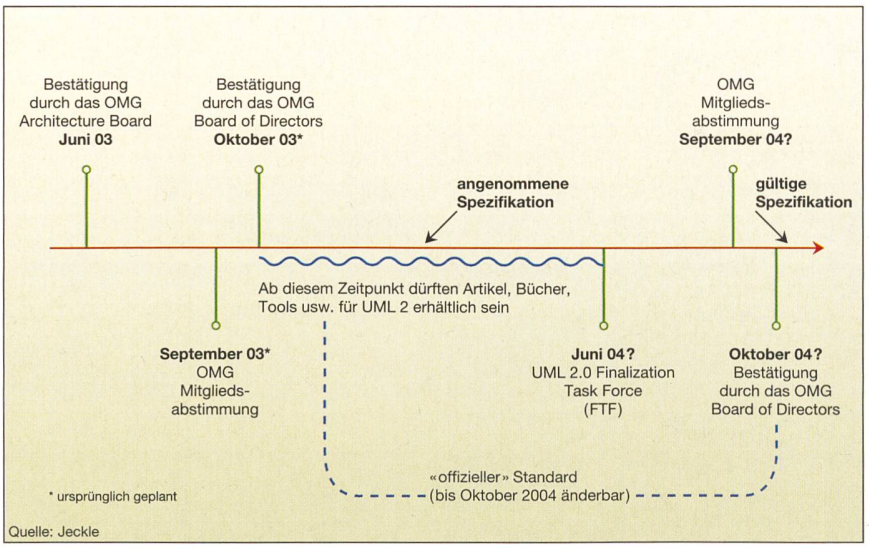

Bild 5 Ein komplexer Annahmeprozess – Der UML-2-Standardisierungsablauf

doch deutlich mächtigeren Notationsmittel der UML 2 verwenden werden, so bleibt der Rückweg versperrt oder ist nur unter Informationsverlust möglich.

- Die aktuelle Version des Superstructure-Dokuments beinhaltet noch einige Kinderkrankheiten (es sind noch Inkonsistenzen und Widersprüche enthalten). Vermutlich wird es hier aber wie bei allen Vorgängerversionen klüger sein, auf ein Buch zum Thema UML 2 [3] zu warten, das die Inhalte des Standards lesergerecht aufbereitet.
- Das grafische Aussehen einiger Notationselemente hat sich verändert, was sicherlich zur Verwirrung führen wird (z.B. sehen Zustände jetzt identisch aus wie die ehemaligen Aktivitäten (jetzt «Aktionen») – somit ist beispielsweise die Aktion des Trinkens nicht vom dadurch erzeugten Zustand der Betrunkenheit zu unterscheiden).
- Eingeführte Bezeichnungen und Namen wurden verändert (z.B. wurde aus dem *Aktivitätsdiagramm* jetzt die *Aktivität* und aus der *Aktivität* nun die *Aktion*).

Umsteigen ja oder nein?

Die Frage «lohnt der Umstieg für mich bzw. mein Unternehmen» lässt sich nicht pauschal beantworten. Für denjenigen, der mit wenig veränderten Diagrammtypen arbeitet und UML nur verwendet, um etwas mehr Transparenz in seine Projekte zu bringen, wird wohl auch weiterhin die «gute alte» (wenn auch mit einigen Fehlern und Unzulänglichkeiten behaftete) UML ausreichend sein. In Grossprojekten mit mehreren beteiligten Interessengruppen dagegen bietet die UML 2 – durch klarere Abgrenzung einzelner Diagrammtypen und sinnvolle Beschränkungen – Vorteile. Treffen einer oder mehrere der folgenden Indikatoren auf das betreffende Projekt bzw. das betreffende Unternehmen zu, sollte man sich einen baldigen Umstieg überlegen:

- Die Modellierung des Systemverhaltens steht im Vordergrund und es werden häufig Zustandsautomaten und Sequenzdiagramme verwendet.
- Man befindet sich im Umfeld technischer Systeme und sucht nach geeigneten Mitteln, um die Kommunikation der Systemkomponenten (z.B. über Ports) darzustellen, oder evtl. auch nach einem Mittel, um das Zeitverhalten des Systems besser modellieren zu können.
- Es werden die Geschäftsprozesse bzw. die Aktivitäten eines Systems mittels Aktivitätsdiagrammen modelliert, und die eingeschränkten Ausdrucksmittel bereiten häufig Ärger.
- Basierend auf einem MDA-Ansatz (Model Driven Architecture) oder in sich schnell ändernden, architektonischen Umfeldern wie EAI (Enterprise Application Integration) werden Code oder Architekturen generiert.

Bevor man umsteigt, sollte man jedoch die UML-2-fähige Version seines Werkzeugs auf Herz und Nieren testen, denn Probleme können hier die Vorteile der neuen Version schnell zunichte machen. Infos über die UML-2-Fähigkeit von Werkzeugen finden sich unter www.uml-glasklar.de.

Mittel- bis langfristig wird sich ein Umstieg auf den neuen Standard nicht vermeiden lassen, da alle neuen Versionen der UML-Werkzeuge die UML 2 ganz oder in Teilen realisieren werden. Für diejenigen, die derzeit keinen grossen Vorteil durch die Neuerungen der UML 2 sehen, kann es sich lohnen, auf die bestätigte und gültige Spezifikation und auf die Werkzeuge, die auf dieser Basis erzeugt wurden, zu warten.

Ein Ausblick

Der weitere Zeitplan der UML-2-Entwicklung ist im Wesentlichen ein Standardisierungs- und Abgleichungsprozess (Bild 5), beginnend im Juni 2003 mit der Bestätigung durch das OMG Architecture

Unified Modeling Language

La nouvelle version du langage de modélisation offre diverses nouveautés

Le Unified Modeling Language (UML) est devenu ces dernières années la notation standard pour l'analyse et les projets de systèmes logiciels. Tandis que les dernières versions de ce langage de modélisation ne présentaient pas de grands changements, la révision pour la version 2.0 attendue pour le printemps 2004 comprend d'importantes nouveautés. En effet, outre l'intégration au langage de modélisation d'éléments de notation attendus depuis longtemps, il y est introduit des diagrammes entièrement nouveaux. Dans quelle mesure UML a-t-il changé? Faut-il maintenant faire passer les projets actuels en UML 2 – et le changement en vaut-il la peine? Et qu'en est-il des nouveaux projets? Autant de questions auxquelles l'article tente de répondre afin d'apporter une aide à la pratique.

Board, über die ein Jahr später geplante Finalization Task Force und hin zur abschliessenden Bestätigung durch das OMG Board of Directors vermutlich im Oktober 2004. [2]

Es bleibt zu hoffen, dass auf Grund des Einflusses der Werkzeughersteller nicht – wie bereits in früheren UML-Versionen geschehen – zugunsten der einfacheren Implementierung gute Ansätze zunichte gemacht werden beziehungsweise Vereinfachungen bereits in die abgegebenen Vorschläge der Konsortien (allen voran U2 Partners) eingeflossen sind. Das wahre Gesicht der neuen UML wird sich wohl frühestens in der Zeit nach der Bestätigung durch das OMG Boards of Directors im Oktober 2004 zeigen, wenn die ersten Tools auf dem Markt erhältlich sind und die UML 2 in ersten Projekten zum Einsatz kommt.

Vielleicht hat die UML-Odyssee mit der bald vorliegenden Version 2.0 ihr Ende gefunden, was allerdings, wenn man sich an der bisherigen Historie orientiert, wohl eher unwahrscheinlich ist. Wann kommt also die UML 2.1?

Referenzen

[1] *Cris Kobryn:* Designing a more agile UML 2.0. Telelogic 2002

[2] *Bran Selic, Brass Bubbles:* An Overview of UML 2.0 (and MDA). IBM 2002

[3] *Mario Jeckle, Chris Rupp, Jürgen Hahn, Barbara Zengler, Stefan Queins:* UML 2 glasklar. München. Carl Hanser Verlag 2003, ISBN 3-446-22575-7. http://www.uml-glasklar.de/

Angaben zu den Autoren

Prof. **Mario Jeckle** ist seit 2003 Professor für Software Engineering an der Fachhochschule Furtwangen. Er ist Ko-Autor der UML 2.0, Mitinitiator des XMI-Ansatzes und vertritt die DaimlerChrysler AG in der Object Managment Group.
www.jeckle.de, mario@jeckle.de

Barbara Zengler ist Inhaberin der Firma Thebit, die Beratung und Dienstleistungen im Umfeld innovativer Techniken anbietet. Ausserdem lektoriert sie die Bücher anerkannter Fachautoren, wie beispielsweise die deutschen Übersetzungen der Werke von Alistar Coburn und Martin Fowler.
www.thebit.biz, bz@barbara-zengler.de

[1] Internet-Seite der Object Management Group: http://www.omg.org/

[2] OMT: Object Modeling Technique

[3] OOSE: Object Oriented Software Engineering

[4] OOD: Object Oriented Design

[5] In den 90-er-Jahren schlossen sich Booch, Jacobson und Rumbaugh zusammen, um ihre Methoden zu vereinen und so die weitere Verbreitung objektorientierter Modellierung zu fördern. Die Ziele der neuen Sprache UML waren die vollständige Modellierung von Systemen mit objektorientierten Techniken, die Klärung von Skalierbarkeitsfragen komplexer, erfolgskritischer Systeme und die Schaffung einer leicht zu verstehenden Modellierungssprache.

[6] XML: eXtensible Markup Language; XMI: XML Metadata Interchange.

[7] J2EE: Java 2 Platform, Enterprise Edition. Komponentenmodell, das die Entwicklung und das Deployment von Unternehmensanwendungen wesentlich beeinflusst hat. J2EE ist ein serverseitiges Komponetenmodell, in das sowohl standardisierte Techniken als auch Programmierschnittstellen (API) eingeschlossen sind. Spezifikation und Referenzimplementierung: http://java.sun.com/j2ee/

[8] MSC: Message Sequence Charts. Verbreiteter grafischer Formalismus zur Beschreibung und Spezifikation von Interaktionen zwischen Systemkomponenten.