Vom Zauber des Modellierens und Simulierens

„Schichten über Schichten, der Lieber wird's schon richten!"
Eine Tour de Force durch UML, Strukturdiagramme und Wetterprognosen.

Nach „Wegen aus der Softwarekrise" (NEW BUSINESS 10/08) agiert Peter Lieber diesmal als „UML-Pfadfinder"

Wenn von „UML Execution" die Rede (bzw. Schreibe) ist, steigen 85 Prozent der LeserInnen „aus". Den restlichen 15 Prozent wiederum ist auch die Kausalkette „Model – Simulate – Generate-Ansatz" noch viel zu trivial. Dabei würde ich so gern vermitteln, wie einfach und vor allem erfolgversprechend im Sinne von „ertragreich" es für viele Unternehmen bzw. Branchen wäre bzw. eigentlich IST, sich ganz kurz mit UML zu beschäftigen. Und mit den zauberhaften Möglichkeiten, die Ihnen LieberLieber damit auf dem „silbernen Tablett" servieren kann.

Unified Modeling Language (UML, engl., vereinheitlichte Modellierungssprache) ist eine (auch über die ISO) standardisierte Sprache für die Modellierung von Software (ISO/IEC 19501). Im Sinne einer Sprache definiert UML die meisten Begriffe, die für die Modellierung wichtig sind, und legt mögliche Beziehungen zwischen diesen Begriffen fest. Die UML definiert weiter grafische Notationen für diese Begriffe und für Modelle von statischen Strukturen und von dynamischen Abläufen.

„Hallo, UML!"

Der erste Kontakt zur UML besteht häufig darin, dass Diagramme im Rahmen von Softwareprojekten zu erstellen, zu verstehen oder zu beurteilen sind:

- Projektauftraggeber prüfen z. B. Anforderungen an ein System, die Wirtschaftsanalytiker in Anwendungsfalldiagrammen der UML festgehalten haben.
- Softwareentwickler realisieren Arbeitsabläufe, die Wirtschaftsanalytiker in Zusammenarbeit mit Fachvertretern in Aktivitätsdiagrammen beschrieben haben.
- Systemingenieure installieren und betreiben Softwaresysteme, basierend auf einem Installationsplan, der als Verteilungsdiagramm vorliegt.

Die grafische Notation ist jedoch nur EIN Aspekt. In erster Linie legt die UML fest, mit welchen Begriffen und ➤➤➤

Die Short Story der UML

Für Leser,
die es „genau wissen" wollen

Die Väter der UML, insbesondere Grady Booch, Ivar Jacobson und James Rumbaugh, auch „Die drei Amigos" genannt, hatten in den 1990er Jahren mehr oder weniger ähnliche Modellierungssprachen entwickelt. Als sie zusammen beim Unternehmen Rational Software beschäftigt waren, sollten die verschiedenen Notationssysteme strukturiert zusammengeführt werden.
Eine Vielzahl von unterschiedlichen Modellierungssprachen hatte direkten oder indirekten Einfluss auf die Konzeption – u. a. OOSE, RDD, OMT, OBA, OODA, SOMA, MOSES, OPEN/OML. Als Resultat dieser Bemühungen entstand die UML. Die Standardisierung, Pflege und Weiterentwicklung der Sprache wurde an die Object Management Group (OMG) übergeben, die UML am 19. 11. 1997 als Standard akzeptierte.
Im September 2000 bat die OMG ihre Mitglieder und weitere interessierte Kreise um Vorschläge für die UML2.
Jahrelang erwies es sich als schwierig, die unterschiedlichen Entwürfe zu einer Spezifikation zu verschmelzen. Kritik wurde laut, dass sich die unterschiedlichen Philosophien in den eingereichten Vorschlägen nur schwerlich würden bereinigen lassen, andererseits reichte im Januar 2003 ein neues Konsortium unter dem Namen 4M einen Vorschlag (UML4MDA) ein, der die Differenzen zwischen den bisherigen Spezifikationen zu überbrücken versuchte. Im September 2004 konnten alle Finalization Task Forces ihre Arbeit beenden, und für UML 2.0 OCL und UML 2.0 Infrastructure lagen damit endgültig abgenommene Dokumente (Final Adopted Specification) vor. Nur bei UML 2.0 Superstructure schien sich dieser letzte Schritt noch etwas zu verzögern: Im März 2005 bot der OMG-Webauftritt weiterhin nur ein temporäres Dokument mit der informellen Bezeichnung UML 2.0 Superstructure FTF convenience document zum Herunterladen an. Im November 2007 legte die OMG die jetzt aktuelle Version UML 2.1.2 vor, und am 21. 10. 2008 wurde die Beta 1 der UML-Version 2.2 durch die OMG veröffentlicht.

Diagramme gefällig?

Die UML2 kennt sechs Strukturdiagramme:

- das Klassendiagramm
- das Kompositionsstrukturdiagramm (auch: Montagediagramm)
- das Komponentendiagramm
- das Verteilungsdiagramm
- das Objektdiagramm und
- das Paketdiagramm

Dazu kommen sieben Verhaltensdiagramme:

- das Aktivitätsdiagramm
- das Anwendungsfalldiagramm (auch Use-Case- oder Nutzfalldiagramm genannt)
- das Interaktionsübersichtsdiagramm
- das Kommunikationsdiagramm
- das Sequenzdiagramm
- das Zeitverlaufsdiagramm und
- das Zustandsdiagramm

Die Grenzen zwischen den 13 Diagrammtypen verlaufen weniger scharf, als diese Klassifizierung vermuten lässt. Die UML2 verbietet nicht, dass ein Diagramm grafische Elemente enthält, die eigentlich zu unterschiedlichen Diagrammtypen gehören.
Es ist sogar denkbar, dass Elemente aus einem Strukturdiagramm und aus einem Verhaltensdiagramm auf dem gleichen Diagramm dargestellt werden, wenn damit eine besonders treffende Aussage zu einem Modell erreicht wird.

Foto: Archiv NEW BUSINESS

Warum Simulationen?

- Eine Untersuchung am realen System wäre zu aufwendig, zu teuer, ethisch nicht vertretbar oder zu gefährlich. Beispiele:
- Fahrsimulator (zu gefährlich in der Realität)
- Flugsimulator zur Pilotenausbildung, Nachstellung kritischer Szenarien (Triebwerksausfall, Notlandung)
- Crashtest (zu gefährlich oder zu aufwendig in der Realität)
- Simulation von Fertigungsanlagen vor einem Umbau (mehrfacher Umbau der Anlage in der Realität wäre zu aufwendig und zu teuer)
- Simulatoren in der chirurgischen Ausbildung (ein Training am Patienten ist in einigen Bereichen ethisch nicht vertretbar)

- Das reale System existiert (noch) nicht. Beispiel: Windkanalexperimente mit Flugzeugmodellen, bevor das Flugzeug gefertigt wird
- Das reale System lässt sich nicht direkt beobachten.
- Systembedingt. Beispiel: Simulation einzelner Moleküle in einer Flüssigkeit, astrophysikalische Prozesse
- Das reale System arbeitet zu schnell. Beispiel: Simulation von Schaltkreisen
- Das reale System arbeitet zu langsam. Beispiel: Simulation geologischer Prozesse

- Für Experimente kann ein Simulationsmodell wesentlich leichter modifiziert werden als das reale System. Beispiel: Modellbau in der Stadtplanung
- Exakte Reproduzierbarkeit der Experimente
- Gefahrlose und kostengünstige Ausbildung. Beispiel: Flugsimulation, Schießausbildung
- Das reale System ist unverstanden oder sehr komplex. Beispiel: Bei der Auswertung wissenschaftlicher Experimente müssen die Ergebnisse per Simulation interpretierbar gemacht werden
- Spiel und Spaß an simulierten Szenarien
- Methode in der Pädagogik. Beispiele: Rollenspiel, Simulationsspiele

➤➤➤ welchen Beziehungen zwischen diesen Begriffen sogenannte Modelle spezifiziert werden – Diagramme der UML zeigen nur eine grafische Sicht auf Ausschnitte dieser Modelle, und die UML schlägt weiter ein Format vor, in dem Modelle und Diagramme zwischen Werkzeugen ausgetauscht werden können.

Alles klar bis hierher?

Man braucht nämlich in den verschiedensten Branchen bzw. Industrien die beiden Stufen „Modellieren“ und „Simulieren“, um zum dritten Schritt „Generieren“ zu kommen. Und Modellierungssprachen wie eben UML (oder auch domänenspezifische Erweiterungen wie z. B. SysML) ermöglichen es Softwareentwicklern, Systemanalytikern oder -architekten, die Anforderungen an ein Organisations- oder ein Softwaresystem sowie dessen Strukturen und inneren Abläufe auf einer höheren Ebene festzulegen. Diese Sprachen versuchen eine Spezifikation für das Management, Benutzer und andere Beteiligte durch Darstellung in Diagrammform möglichst verständlich zu machen. (Ok, einfach ist es nicht …)

Obwohl ein weiteres Ziel der Modellierungssprachen die Programmierung ohne Programmierer ist, kommen Programmierer immer dann ins Spiel, wenn die Spezifikation der Anforderungen abgeschlossen ist. Denn hier geht es dann schon um Simulationen, also um eine Analyse von Systemen, die für die theoretische oder formelmäßige Behandlung zu kompliziert sind. Bei der Simulation werden Experimente an einem Modell durchgeführt, um Erkenntnisse über das reale System zu gewinnen, die dann interpretiert und auf das zu simulierende System übertragen werden.

Ist ein vorhandenes Modell geeignet, um Aussagen über die zu lösende Problemstellung zu machen, müssen lediglich die Parameter des Modells entweder hinsichtlich der Istsituation oder einer gewünschten Zielsituation eingestellt und gegebenenfalls geeignet variiert werden. Simulationen werden für viele Problemstellungen in der Praxis eingesetzt – etwa für Strömungs- oder Verkehrssimulationen oder für Wettervorhersagen.

Und wenn die Wetterprognose NICHT STIMMT?

Ein Auto-Crashtest z. B. ist eine Simulation für eine reale Verkehrssituation, in der ein Auto in einen Verkehrsunfall verwickelt ist. Dabei werden die Vorgeschichte des Unfalls, die Verkehrssituation und die genaue Beschaffenheit des Unfallgegners stark vereinfacht. Auch sind keine Personen in den simulierten Unfall verwickelt, stattdessen werden Crashtest-Dummies eingesetzt, die mit realen Menschen gewisse mechanische Eigenschaften gemeinsam haben. Ein Simulationsmodell hat also nur ganz bestimmte Aspekte mit einem realen Unfall gemeinsam.

Jeglicher Form von Simulation sind auch Grenzen gesetzt, die man stets beachten muss. Die erste Grenze folgt aus der Begrenztheit der Mittel, d. h. der Endlichkeit von Energie (z. B. auch Rechenkapazität), Zeit und nicht zuletzt Geld. Eine Simulation muss also auch wirtschaftlich gesehen Sinn ergeben. Deshalb muss ein Modell möglichst einfach sein. Das wiederum bedeutet, dass auch die verwendeten Modelle oft eine grobe Vereinfachung der Realität darstellen. Diese Vereinfachungen beeinträchtigen naturgemäß auch die Genauigkeit der Simulationsergebnisse. Die zweite Grenze folgt daraus: Ein Modell liefert nur in einem bestimmten Kontext Ergebnisse, die sich auf die Realität übertragen lassen. In anderen Parameterbereichen können die Resultate schlichtweg falsch sein. Daher ist die Verifikation der Modelle für den jeweiligen Anwendungsfall ein wichtiger Bestandteil der Simulationstechnik. Weitere „Grenzen“ sind Ungenauigkeiten der Ausgangsdaten (z. B. Messfehler) sowie subjektive Hindernisse (z. B. mangelnder Informationsfluss über Produktionsfehler).

Und spätestens an diesem Punkt sollten Sie sich GANZ SUBJEKTIV dafür entscheiden, uns ganz persönlich zu kontaktieren, um den RICHTIGEN Informationsfluss zu gewährleisten! <<

Kontakt

Die LieberLieber Software GmbH zählt wie z. B. SparxSystems Software GmbH (Central Europe) zur Firmengruppe von Peter Lieber und agiert als Microsoft Gold Partner mit Windows- und Webkompetenz u. a. in den Bereichen Web, Sharepoint, Intranet, Windows, Office-Erweiterungen sowie Robotersteuerung. Bemerkenswert sind weiters die speziellen Produkte für Behinderte (z. B. im Fall von Sprachstörungen), Scheduled Mediaplayer für Mediawalls, elektronische Bilderrahmen u. v. a. m.

peter.lieber@lieberlieber.com
www.lieberlieber.com
www.sparxsystems.at