Die Unified Modeling Language

Die UML (hier in der Version 0.9) ist ein Satz von Notationen zur Beschreibung objektorientierter Softwaresysteme. Ihre Autoren sind:

Slide 1

Grady Booch: „*Object oriented design and applications*“

James Rumbaugh: Co-Autor der *Object Modelling Technique*

Ivar Jacobson: *Use-Cases*

(z.Zt. sind alle drei bei *Rational, Inc.*, `http://www.rational.com` beschäftigt)

Aufgaben und Ziele der UML

Die UML soll nach den Wünschen der Autoren eine Reihe von Aufgaben und Zielen verfolgen:

Slide 2

- Bereitstellung einer universellen Beschreibungssprache für alle Arten objektorientierter Softwaresysteme
- Vereinigung der bekanntesten Beschreibungsnotationen
- Setzen des Schwerpunkts auf die Produkte des Software-Engineerings und nicht auf den Prozeß

Es lassen sich jedoch einige andere interessante Entwicklungen anhand der UML als Vertreterin der Methoden/Notationen der dritten Generation beobachten, dazu später mehr.

Hintergrund

Zur Einordnung der UML ist eine Betrachtung des Entstehungshintergrunds und der beteiligten Personen interessant:

Slide 3

- die OMT stellte zu Anfang einen Großteil der grundlegenden Notationen
- die UML wird der OMG als Antwort auf einen Aufruf zur Erstellung einer standardisierten objektorientierten Entwurfsmethode vorgelegt
- Einflüsse von Ada und C++ sind unverkennbar

Richtlinien bei der Entwicklung der UML

Bei der Entwicklung der (hauptsächlich graphischen) Notation sollten u.a. folgende Prinzipien erfüllt werden:

Slide 4

Einfachheit: Verwendung weniger Konzepte und Symbole

Priorisierung: Einfache Modellierung häufiger Probleme, mehr Aufwand bei ungewöhnlichen Situationen

Konsistenz: Verwendung gleicher Konzepte in allen Bereichen der Notation

Richtlinien bei der Entwicklung der UML

Slide 5

Orthogonalität: Verwendung unterschiedlicher Notationen für unterschiedliche Konzepte

Schichtenarchitektur: Erweiterbarkeit der Notationen um fortgeschrittene Konzepte

Stabilität: Adaption bekannter Konzepte und Notationen

Druck- und Zeichenbarkeit: Eignung sowohl für manuelle als auch werkzeuggestützte Erstellung

Das Metamodell

Slide 6

Die UML wird formal in einem *Metamodell* beschrieben. Dabei werden die Elemente der UML selbst benutzt, um das Metamodell zu beschreiben. Ziele sind:

- Beschreibung der Semantik der UML
- Bereitstellung einer formalen Basis
- Unterstützung bei der Erstellung von Tools
- Definition eines Austauschformats für Modelle

Metamodell und Zielsprache

Die UML ist unabhängig von der Zielprogrammiersprache einsetzbar. Um Aspekte der Zielsprache (z.B. C++, SmallTalk, Ada . . .) in Modellen aufnehmen zu können, werden im Metamodell Annotationen der einzelnen Elemente zugelassen. Sie werden als **uninterpretierte Strings** verwendet.

Slide 7

Bestandteile des Metamodells

Das Metamodell hat auf der abstrakten Ebene folgende Bestandteile:

Element: Oberbegriff für alle Elemente eines Modells

SubjectElement: beschreibt logische und physikalische Eigenschaften

Slide 8 **ViewElement:** Umsetzung der logischen und physikalischen Eigenschaften in eine graphische Form

Auf den konkreten Ebenen werden die einzelnen Diagrammtypen beschrieben.

Diagrammtypen

Basisdiagrammtypen der UML sind:

- *Class Diagrams*
- *Sequence Diagrams*

Slide 9

- *Collaboration Diagrams*
- *State Charts*

Erweiterte Modellierungsmöglichkeiten liefern *Use Case Diagrams*, *Component Diagrams*, *Deployment Diagrams* und *Operation Specifications*

Übergeordnete Konzepte

Als globale Konzepte sind in der UML

- *Packages* und
- *Stereotypes*

Slide 10

verfügbar

Class Diagrams

Als wichtigstes Beschreibungselement dienen die *Class Diagrams*. Sie beschreiben sowohl Klassen als auch Objekte (eine strikte Trennung ist nicht vorgesehen, vielmehr wird die Ansicht vertreten, Klassen können auch Objekte sein und umgekehrt)
Die Notation in den Klassendiagrammen lehnt sich dabei stark an die *OMT* an,
Slide 11 auch sind Parallelen zu ER-Diagrammen unverkennbar.

Elemente in Klassendiagrammen

Elemente von Klassendiagrammen sind:

- Klassen mit *Attributen* und *Operationen*
- Generische Klassen (*Templates*)

Slide 12

- Zusammengesetzte Klassen (*Composites*)
- Utility-Klassen (Sammlungen von Daten oder Funktionen)
- Objekte

Beziehungen in Klassendiagrammen

Klassendiagramme können eine Vielzahl von **Beziehungen** zwischen den Elementen beschreiben. Zu den Assoziationen können Kardinalitäten der beteiligten Elemente angegeben werden.
Die Beteiligung eines Elements an einer Assoziation wird dabei als **Rolle** bezeichnet.

Slide 13 Zwischen einzelnen Assoziationen können wiederum zwei Beziehungen bestehen: *Einschränkung* und *Ableitung* von Werten

Beziehungen in Klassendiagrammen

Mögliche Beziehungen sind:

- einfache (benannte) Assoziationen
- Assoziationen mit angefügten Attributen oder Klassen

Slide 14

- Qualifizierte Assoziationen
- Aggregationen
- Assoziationen zwischen drei und mehr Elementen
- Navigationsassoziationen
- Vererbung

Assoziationen und Attribute

Attribute werden von Assoziationen unterschieden:

Assoziation: Beschreibt Beziehungen, bei denen beteiligte Klassen und Objekte von anderen Klassen und Objekten benutzt werden können

Slide 15

Attribut: Beschreibt einen *privaten* Bestandteil einer Klasse oder eines Objekts, welcher von außen nicht sichtbar bzw. modifizierbar ist

Sequence Diagrams

Mit *Sequence Diagrams* werden Szenarien beschrieben, bei denen mehrere Objekte kommunizieren. Der Schwerpunkt liegt hier auf der Betrachtung zeitlicher Aspekte. Dementsprechend werden folgende Elemente in Sequence Diagrams verwendet:

Slide 16

- beteiligte Objekte (von der Erzeugung bis zur Zerstörung)
- eine Zeitachse (die für jedes Objekt eingetragen wird)
- Funktionsaufrufe oder Nachrichten (können mit Bedingungen versehen werden)
- Zeitmarkierungen
- Ausdrücke mit Zeitmarkierungen

Collaboration Diagrams

Zur Beschreibung von dynamischen Beziehungen zwischen Objekten werden *Collaboration Diagrams* verwendet. Sie bestehen aus Objekten und Verbindungen zwischen den Objekten.
Collaboration Diagrams stellen den Systemzustand vor und nach einer Operation dar, wobei die Unterschiede zwischen den Systemzuständen mittels Annotationen im Diagramm kenntlich gemacht werden.

Slide 17

Verbindungen in Collaboration Diagrams

Verbindungen können einerseits permanent sein (d.h. Assoziationen aus den Klassendiagrammen), andererseits sind auch temporäre Verbindungen vorgesehen. Mögliche Verbindungen sind:

Slide 18

- Assoziation (A)
- Attribut (*Object Field*, F)
- globale Variable (G)
- lokale Variable (L)
- Funktionsparameter (P)
- Selbstreferenz (S)

Verbindungen in Collaboration Diagrams

Verbindungen können mit Annotationen versehen werden:

Slide 19

- Sequenznummern zur Angabe von Operationen, die vorher stattgefunden haben müssen
- Aufrufsequenznummern geben an, auf welcher Ebene der Aufrufhierarchie ein Funktionsaufruf erfolgt
- Rückgabewerte
- Name der Nachricht oder der aufgerufenen Funktion
- Parameter
- Bedingungen

Nebenläufigkeit

Nebenläufigkeit wird durch verzweigende Verbindungen dargestellt. Unabhängige Objekte werden dabei mit Hilfe von *Composites* gebildet.
Die unterschiedlichen Threads werden dabei mit Annotationen zu den Aufrufsequenznummern unterschieden.

Slide 20

State Diagrams

State Diagrams orientieren sich stark an Harels *State Charts*. Zu bermerken ist aber, daß State Diagrams neben dem *Broadcast*-Mechanismus für das Versenden von Ausgaben an andere Objekte auch vorsehen, daß das Zielobjekt direkt angegeben wird.

Slide 21

Use Case Models

Als fortgeschrittene Notation (bzw. Methode) werden *Use Case Models* verwendet. Die ermöglichen es, ein System aus Anwendersicht nach Funktionalitäten zu modellieren.
(Informationen zu Use Cases finden sich in den Unterlagen zu *Modellierung I*)

Slide 22

Component Diagrams

Module und andere Softwarekomponenten werden in *Component Diagrams* dokumentiert. Bestandteile sind:

- Module
- Abhängigkeiten (*Benutzt-Beziehung* oder Übersetzungabhängigkeiten)

Slide 23

Deployment Diagrams

Ähnlich wie *Component Diagrams* werden *Deployment Diagrams* zu Modellierung physikalischer Eigenschaften benutzt. Sie beschreiben die Verteilung von Prozessen auf Rechnern.
Als Ausführungsplattform (*Node* kommen dabei neben Rechnern aber auch Drucker, Controller usw. in Betracht.

Slide 24

Operation Specification

Funktionen und Operationen werden in *Operation Specifications* festgehalten. Diese enthalten folgende Elemente:

Slide 25

- Aufgabenbeschreibung
- Vor- und Nachbedingungen (auch als *Collaboration Diagrams* formulierbar
- Ein- und Ausgabewerte
- modifizierte Objekte

Packages

Packages erlauben es, als allgemeines Notationselement alle Arten von Diagrammen und Modellen zu übergeordneten Einheiten zusammenzufassen. Zwischen Packages können Abhängigkeiten bestehen, so daß z.B. eine Package auf andere Packages zugreift. Außerdem lassen sich Packages schachteln.
Der Name einer Package kann zur Unterscheidung der Herkunft einer Klasse oder eines anderen Elements dienen (vgl. auch Präfixnotation in Modula-2)

Slide 26

Stereotypes

Ein wichtiges, allgemein einsetzbares Element zur Unterstützung bei der Modellierung sind *Stereotypes.* Sie bieten einen Mechanismus zur Kategorisierung jeglicher Elemente der UML.
Beispiele sind z.B. *Interface*, *Handler*, *Exception* u.a., welche oft während der Modellierung als Begriffe verwendet werden und in der UML als fest vorgesehene
Slide 27 Annotation einsetzbar sind. Auch ist mit Stereotypes die Beschreibung einer*Metaebene* möglich.

Zur Vereinfachung definiert die UML einen Satz von Kern-Stereotypes, die die gängigsten Fälle abdecken.

Interfaces

Interfaces sind ein Beschreibungsmittel, welche mit *Stereotypes* realisiert werden. Interfaces werden als Beziehungen zwischen Objekten modelliert, Sie werden mit dem Stereotype *conforms* und dem Namen des Interfaces versehen.
Für die Beschreibung kann jedoch auch eine Kurzform gewählt werden, bei der ein Objekt mit einer Reihe von Interfaces versehen wird (wobei allerdings nicht
Slide 28 unterschieden wird, welche Operationen und Attribute an einem Interface teilnehmen).

Verteilte Systeme

Die Verteilung von beliebigen Elementen wird über die vordefinierte Eigenschaft *location* beschrieben.
Diese Eigenschaft kann auch zur Migration von Objekten benutzt werden, indem in einem *Collaboration Diagram* eine Verbindung zwischen Ausgangs- und Zielobjekt mit dem Stereotype *becomes* angegeben wird und die Eigenschaft
Slide 29 *location* entsprechend geändert wird.

Schlußbetrachtung

Die UML zeigt einige interessante Aspekte auf:

- sie verzichtet ganz auf eine Beschreibung des Vorgehensmodells
- die einzelnen Notationen werden nicht mehr untereinander abgeglichen

Slide 30

- es existieren noch einige Einflüsse von C++, die aber teilweise reduziert werden.
- auch sind Einflüsse aus der Datenbankwelt unverkennbar
- allgemeine Konzepte (*Meta-Konzepte*) können notationsübergreifend benutzt werden