

UML Unified Modeling Language

+ [andere TechDocs](#)
+ [Vorgehensmodelle](#)
+ [Homepage Torsten Horn](#)
+

UML (Unified Modeling Language) ist ein Standard der OMG (<http://www.omg.org/uml>) und ist keine Methode, sondern definiert eine Notation und Semantik zur Visualisierung, Konstruktion und Dokumentation von Modellen für die Geschäftsprozessmodellierung und für die objektorientierte Softwareentwicklung.

MDA (Model Driven Architecture) ist ebenfalls ein Standard der OMG (<http://www.omg.org/mda>) und definiert eine Vorgehensweise beim Softwareentwicklungsprozess unter Verwendung von UML. MDA unterscheidet zwischen PIM (Platform Independent Model), PSM (Platform Specific Model) und Code und bietet eine Grundlage für automatische Code-Generierung.

Inhalt

1. [Übersicht zu den 13 UML-Diagrammtypen](#)
2. [UML-Use-Case-Diagramm \(Anwendungsfalldiagramm, Use Case Diagram\)](#)
 1. [Begriffe](#)
 2. [Use-Case-Diagrammbeispiel](#)
3. [UML-Aktivitätsdiagramm \(Activity Diagram\)](#)
 1. [Begriffe](#)
 2. [Aktivitätsdiagrammbeispiel](#)
4. [UML-Sequenzdiagramm \(Sequence Diagram\)](#)
 1. [Definition](#)
 2. [Sequenzdiagrammbeispiel](#)
5. [UML-Klassendiagramm \(Class Diagram\)](#)

1. [Begriffe, Definitionen und Prinzipien der Objektorientierung](#)
2. [Klassen-Stereotypen](#)
3. [Einige UML-Notationen](#)
6. **[MDA](#)**
 1. [Definition](#)
 2. [Begriffe](#)
 3. [Mögliche Anforderungen an UML-/MDA-Tools zur automatischen Codegenerierung](#)
7. **[Links auf weiterführende Informationen](#)**

Übersicht zu den 13 UML-Diagrammtypen

- Behavior Diagrams (Verhaltensdiagramme)
 - Use Case Diagram ([Use-Case-Diagramm, Anwendungsfalldiagramm](#))
(stellt Beziehungen zwischen Akteuren und Anwendungsfällen dar)
 - Activity Diagram ([Aktivitätsdiagramm](#))
(beschreibt Ablaufmöglichkeiten, die aus einzelnen Aktivitäten/Schritten bestehen)
 - Statechart Diagram (Zustandsdiagramm, Zustandsautomat)
(zeigt eine Folge von Zuständen eines Objekts)
 - Sequence Diagram ([Sequenzdiagramm](#))
(wichtigstes Interaktionsdiagramm: zeigt den zeitlichen Ablauf von Nachrichten zwischen Objekten)
 - Communication Diagram (Kommunikationsdiagramm) (früher Kollaborationsdiagramm)
(Interaktionsdiagramm: zeigt Beziehungen und Interaktionen zwischen Objekten)
 - Timing Diagram (Timingdiagramm, Zeitverlaufsdiagramm)
(Interaktionsdiagramm mit Zeitverlaufskurven von Zuständen)

- Interaction Overview Diagram (Interaktionsübersichtsdiagramm)
(Interaktionsdiagramm zur Übersicht über Abfolgen von Interaktionen, ähnlich Aktivitätsdiagramm)
- Structural Diagrams (Strukturdiagramme)
 - Class Diagram ([Klassendiagramm](#))
(wichtigstes Diagramm: Klassen und ihre Beziehungen untereinander)
 - Package Diagram (Paketdiagramm)
(Gliedert Softwaresysteme in Untereinheiten)
 - Object Diagram (Objektdiagramm)
(Objekte, Assoziationen und Attributwerte zu einem bestimmten Zeitpunkt während Laufzeit)
 - Composite Structure Diagram (Kompositionsstrukturdiagramm)
(Abbildung innerer Zusammenhänge einer komplexen Systemarchitektur, Darstellung von Design Patterns)
 - Component Diagram (Komponentendiagramm)
(Komponenten und ihre Beziehungen und Schnittstellen)
 - Deployment Diagram (Verteilungsdiagramm)
(Einsatzdiagramm, Knotendiagramm, Laufzeitumfeld)

UML-Use-Case-Diagramm (Anwendungsfalldiagramm, Use Case Diagram)

Begriffe

- Use-Case-Diagramm (Anwendungsfalldiagramm, Use Case Diagram):
Ein Use-Case-Diagramm stellt Beziehungen zwischen Akteuren und Anwendungsfällen aus Sicht der Akteure/Stakeholder dar. Use-Case-Diagramme sind vorwiegend ein Hilfsmittel zur Anforderungsermittlung und dienen weniger der Verhaltensbeschreibung und dem Systemdesign. Ein Anwendungsfall beschreibt einen Ablauf, aber ein Anwendungsfalldiagramm ist keine Ablaufbeschreibung, es wird keine Ablaufreihenfolge definiert. Dafür gibt es andere UML-Verhaltensdiagramme (z.B. Aktivitäts-, Zustands-, Sequenz- und Kommunikationsdiagramm).
- Akteur (Actor, Stakeholder, «actor»):
Ein Akteur ist eine an Anwendungsfällen beteiligte aber außerhalb des zu realisierenden Systems agierende Einheit (Mensch oder technisches System). Menschliche Akteure werden nicht personifiziert, sondern nur die Rolle im Kontext des Anwendungsfalls ist relevant.

- **Anwendungsfall (Use Case):**

Ein Anwendungsfall beschreibt eine zeitlich ununterbrochene Interaktion (einen einzelnen Arbeitsgang) eines oder mehrerer Akteure mit einem System aus der Sicht des Anwenders.

- **Geschäftsanwendungsfall (Geschäftsfall, Business Use Case, «business»):**

Ein Geschäftsanwendungsfall beschreibt einen Anwendungsfall in abstrakter fachlicher Form aus Sicht des Anwenders. Aus einem fachlichen Geschäftsanwendungsfall (z.B. reservieren) können sich mehrere konkrete Systemanwendungsfälle ergeben (z.B. telefonisch, online, FAX).



- **Geschäftsprozess («workflow»):**

Ein Geschäftsprozess kann aus mehreren Anwendungsfällen bestehen und stellt eine Zusammenfassung von fachlich zusammenhängenden Aktivitäten dar, die durchgeführt werden, um einen Geschäftsvorfall ergebnisorientiert zu bearbeiten. Geschäftsprozesse können in UML-Aktivitätsdiagrammen oder ARIS-EPKs (ereignisgesteuerte Prozessketten) visualisiert werden.

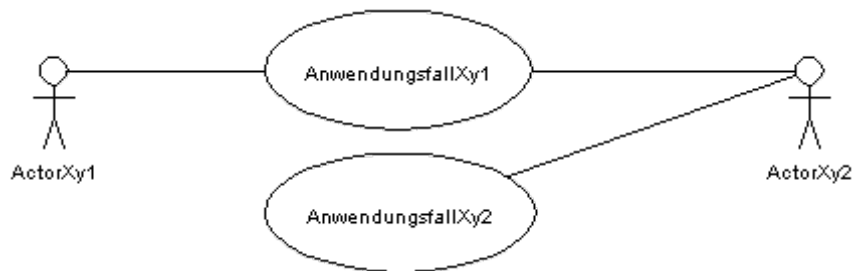
- **Geschäftsvorfall:**

Ein Geschäftsvorfall (z.B. Antrag) entsteht durch ein Ereignis (z.B. Antragseingang) und hat fachliche Ergebnisse (z.B. Vertrag).

- **Anwendungsfallbeziehungen:**

- Vererbung, dargestellt mit dem [Vererbungspeilsymbol](#).
- «include»: anderer Anwendungsfall innerhalb des Anwendungsfalls, dargestellt mit gestricheltem Pfeil.
- «extend»: Anwendungsfall wird unter bestimmten Bedingungen an einer bestimmten Stelle (Erweiterungspunkt, extension point) durch einen anderen Anwendungsfall erweitert, dargestellt mit gestricheltem Pfeil.

Use-Case-Diagrammbeispiel

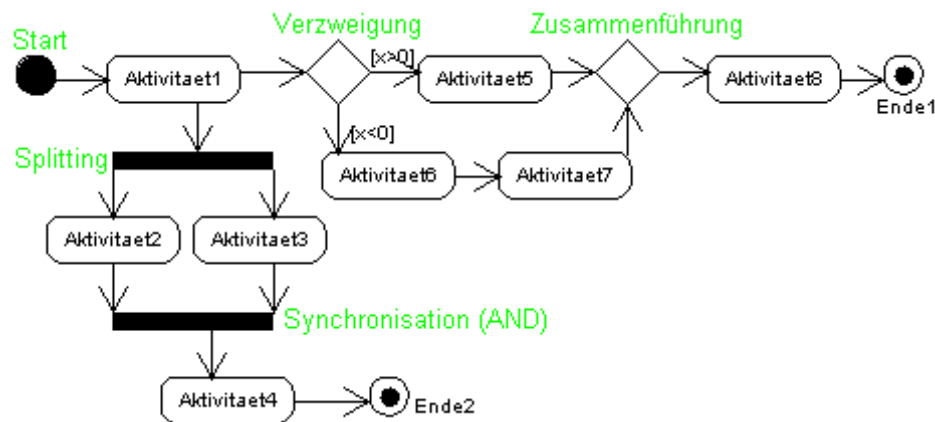


UML-Aktivitätsdiagramm (Activity Diagram)

Begriffe

- **Aktivitätsdiagramm:**
Aktivitätsdiagramme beschreiben Ablaufmöglichkeiten, die aus einzelnen Aktivitäten (Schritten) bestehen.
- **Transition:**
Eine Transition ist ein Zustandsübergang. Die Aktivitäten beginnen mit einer eingehenden Transition und haben eine oder mehrere ausgehende Transitionen.
- **Verzweigung (Branch, Decision):**
Verzweigungen, die auf Grund einer Bedingung stattfinden, können entweder mit dem Rautensymbol symbolisiert werden (siehe Aktivitätsdiagrammbeispiel) oder die verschiedenen ausgehenden Transitionen starten direkt am Aktivitätssymbol. In beiden Varianten werden die Verzweigungsbedingungen in eckigen Klammern angegeben.
- **Zusammenführung:**
Sobald eine der eingehenden Transitionen eingeht, geht der Aktivitätsfluss weiter (anders als bei der Synchronisation).
- **Splitting (Fork):**
Der Aktivitätsfluss wird auf mehrere Aktivitäten verteilt (anders als bei der Verzweigung).
- **Synchronisation (AND) (Join):**
Der Aktivitätsfluss geht erst dann weiter, wenn alle eingehenden Transitionen vorliegen (anders als bei der Zusammenführung).
- **Signale senden und empfangen:**
Beim Übergang von einer Aktivität zur nächsten können Signale an Objekte gesendet oder empfangen werden.

Aktivitätsdiagrammbeispiel



UML-Sequenzdiagramm (Sequence Diagram)

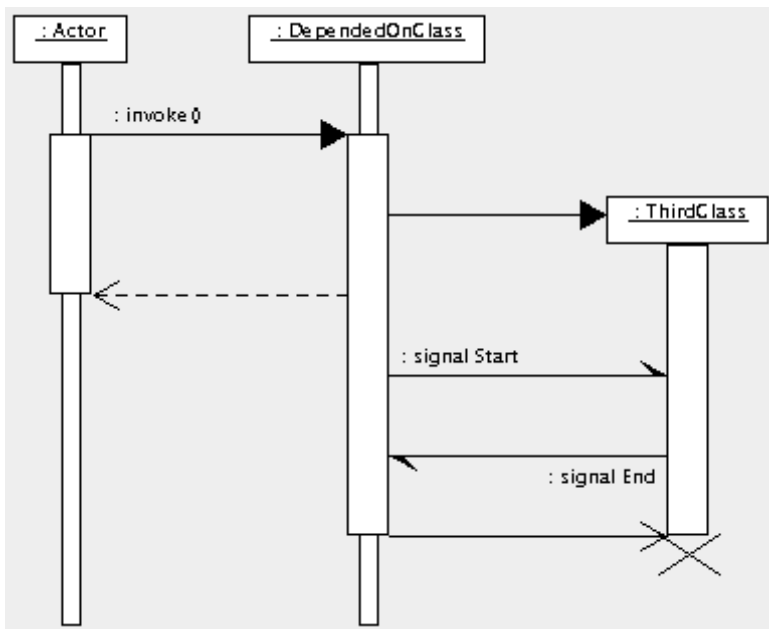
Definition

Sequenzdiagramme sind die wichtigsten Interaktionsdiagramme und zeigen den zeitlichen Ablauf einer Reihe von Nachrichten (Methodenaufrufen) zwischen bestimmten Objekten in einer zeitlich begrenzten Situation. Dabei kann auch das Erzeugen und Entfernen von Objekten enthalten sein.

Sequenzdiagramme legen den Schwerpunkt auf den zeitlichen Nachrichtenverlauf, während andere Diagramme (z.B. Kommunikationsdiagramm) die Zusammenarbeit der Objekte verdeutlichen.

Manchmal verwendete Abkürzungen: 'ref' = Referenz auf bereits definierte Sequenz, 'alt' = Auswahl unter Alternativen, 'opt' = optionale Ausführung, 'par' = parallele Ausführung, 'loop' = iterative Schleife, '{...}' = Constraint.

Sequenzdiagrammbeispiel



UML-Klassendiagramm (Class Diagram)

Begriffe, Definitionen und Prinzipien der Objektorientierung

- **Kapselung:**
Klassen fassen Attribute und Operationen zusammen.
- **Objekte aus Klassen:**
Klassen (Class) trennen die generelle Beschreibung der Struktur und des Verhaltens von den tatsächlich instanziierten Objekten (Exemplar, Object, Instance).
- **Objekt-Identität:**
Auch von der gleichen Klasse instanziierte und die gleichen Attributwerte enthaltenden Objekte sind unterscheidbare individuelle Objekte.
- **Kohärenz:**
Jede Klasse sollte möglichst nur genau einen sachlogischen Aspekt widerspiegeln und dafür verantwortlich sein. Eigenschaften sollten nicht wegen Optimierungen und Redundanzfreiheit einer Klasse zugeordnet werden, wenn sachlogisch/fachlich eine andere Klasse zuständig ist.
- **Vererbung, Spezialisierung:**
Klassen können hierarchisch strukturiert werden und Eigenschaften an Unterklassen vererben (Inheritance). Entsprechend einer Spezialisierung erben Unterklassen alle Eigenschaften der Oberklasse und fügen weitere Eigenschaften hinzu. Unterklassen können die geerbten Eigenschaften der Oberklasse überschreiben und erweitern, jedoch nicht eliminieren oder unterdrücken.
- **Substitution:**
Objekte von Unterklassen können jederzeit anstelle von Objekten ihrer Oberklasse eingesetzt werden.
- **Statische Polymorphie, Überladen:**
Es können mehrere gleichnamige Operationen (Methoden) innerhalb einer Klasse definiert werden, wenn sich deren Signatur unterscheidet, also wenn die verwendeten Methodenparameter von unterschiedlichen Typen sind.
- **Dynamische Polymorphie, Überschreiben:**
Wird eine Methode in mehreren von der gleichen Vorfahrenklasse abgeleiteten Klassen überschrieben und unterschiedlich implementiert und auf diese Methoden über eine Variable vom Typ der Vorfahrenklasse zugegriffen, nennt man das dynamische Polymorphie.
- **Persistenz, Transienz:**
Persistente Objekte sind serialisierbar und werden langfristig gespeichert (z.B. im Dateisystem oder in einer Datenbank).
Transiente Objekte werden nur vorübergehend benötigt und nicht gespeichert und gehen bei Programmende verloren.
- **Stereotyp:**
Stereotypen erweitern vorhandene Modellelemente zu neuen Modellelementen des UML-Metamodells. Sie definieren oft Verwendungszusammenhänge.
Unterschieden werden:
"dekorative Stereotypen" (zur Gestaltung),

"deskriptive Stereotypen" (Verwendungszusammenhangsbeschreibung, Kommentierung, z.B. «entity», «control»),
"restriktive Stereotypen" (formale Einschränkungen, Restriktionen, Eigenschaftsexistenz, z.B. «interface») und
"redefinierende Stereotypen" (Metamodell-Modifikationen).

- **Design Patterns, Entwurfsmuster:**

Entwurfsmuster stellen standardisierte Lösungsvorschläge für immer wiederkehrende Probleme dar. Sie sind unabhängig von Programmiersprachen und liegen nicht fertig codiert vor, sondern als Beschreibung.

Architekturmuster definieren grobe Strukturen wie z.B. eine Three-Tier-Architektur. Die meisten Entwurfsmuster befassen sich mit kleineren Problemen, zum Beispiel: Adapter (erweitert um Schnittstelle), Bridge (Brücke, trennt Schnittstelle von Implementierung), Decorator (Dekorierer, ändert dynamisch Eigenschaften), Facade (Fassade, vereinfachte gemeinsame Schnittstelle), Singleton (nur eine globale Instanz), Memento (Undo-Funktion), Observer (Beobachter, Benachrichtigung bei Zustandsänderung), Composite (Kompositum, baumartige Aggregation).

Weiteres zu Mustern und Patterns siehe [techdocs/sw-patterns.htm](https://techdocs.sw-patterns.htm).

- **Subsystem, Paket, Komponente:**

Subsysteme sind Teile eines Gesamtsystems, die nach außen durch eine Schnittstelle definiert sind.

Pakete sind dagegen nur lose Sammlungen von Modellelementen innerhalb eines Namensraums.

Komponenten andererseits sind prinzipiell austauschbare Softwareeinheiten, kapseln komplexes Verhalten mit hoher fachlicher Kohärenz, werden oft wie Klassen instanziiert und stellen definierte Schnittstellen (Interfaces) bereit (oft "Factory Services", z.B. `findByName()` und `findByKey()`, "Observer Services", z.B. `addChangeListener()` und `addVetoableChangeListener()` sowie "Object Services" für fachliche Operationen).

- **OCL (Object Constraint Language):**

Syntax zur Beschreibung von Zusicherungen, Bedingungen, Integritätsregeln und zulässige Wertemengen mit Hilfe von Preconditions ("pre"), Postconditions ("post") und Invarianten ("inv") und meistens eingeleitet über das Schlüsselwort "context" ("context ... inv:" kann auch weggelassen werden, wenn der Kontextgegenstand unterstrichen wird). Beispiele:

```
context Modellelement inv constraintName:
```

```
    Zusicherung
```

```
context Person inv Volljaehrigkeit:
```

```
    self.alter >= 18
```

```
Person
```

```
    self.alter >= 18
```

```
context Typ::operation(p1 : type1): ReturnType
```




```
    pre: parameterOk: p1 = ...
```

```
    post: resultOk:    result = ...
```

```
Projekt
```

```
    self.projektmitglieder->includes(self.projektleiter)
```

Klassen-Stereotypen

«interface»	keine Attribute nur abstrakte Operationen nicht instanzierbar	Schnittstellenklassen sind abstrakte Klassen zur Definition funktionaler Schnittstellen. Sie erleichtern arbeitsteilige Softwareentwicklung ("Design by Contract"). Schnittstellenklassen können von anderen Schnittstellenklassen erben («extend») und in konkreten Klassen realisiert werden (Realisierungsbeziehung, «realize»).	
«type»	meistens Attribute meistens abstrakte Operationen Assoziationen zu anderen Typen	Typen sind abstrakte Spezifikationen (ähnlich Schnittstellen) von strukturellen Schnittstellen.	
«entity»	viele Attribute viele primitive Operationen (Getter/Setter) wenige komplexe Operationen	Entitätsklassen repräsentieren fachlichen Sachverhalt oder Realweltgegenstand (Vertrag, Kunde, Adresse).	
«control»	wenige Attribute transient, kurze Lebensdauer komplexe Operationen	Steuerungsklassen dienen Ablauf-, Steuerungs- und Berechnungsvorgängen, meistens stark klassenübergreifend.	
«boundary»	keine eigenen Operationen delegieren Operationsaufrufe weiter keine fachliche Logik fast nur abgeleitete oder ableitbare Attribute keine Persistenz, keine Zustände oft Singletons	Schnittstellenobjekte bilden eine Zusammenstellung von Eigenschaften anderer Objekte, zum Beispiel zur Entkopplung im Sinne von Fassaden.	
«primitive»	elementare Klassen und Standardklassen wenige Attribute einfache Operationen	Primitive Klassen werden in Deklarationen von Attributen verwendet. Sie werden nicht als Klasse im Klassendiagramm dargestellt.	
«enumeration»	ähnliche wie bei primitiven Klassen fast nur in Deklarationen für Attribute	Enumerationen sind aufzählbare Wertemengen. Können Werte für Auswahllisten repräsentieren.	
«structure»	ausschließlich Attributdefinitionen keine Operationen	Datenstrukturen werden normalerweise nur zum Datenaustausch mit anderen Systemen verwendet.	
«utility»	Klassenattribute Klassenoperationen	Sammlungen von globalen Variablen und Funktionen.	

Einige UML-Notationen



Paketname::Klassenname {Eigenschaftswerte}
± Attributname : Attributtyp = Initialwert {Zusicherung}
± Methodenname(Parameter:Typ) {Eigenschaft}

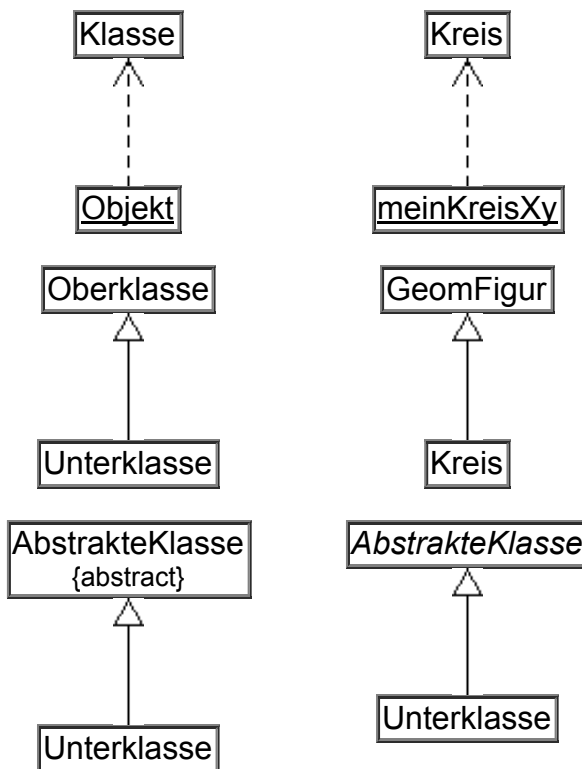
KreisBeispiel
radius {radius>0}
mittelpunkt : Point = (20,30)
setRadius(rad)
setPosition(pos: Point)

Drei Rubriken:

- 1.) Klassenname (eventuell mit Zusätzen)
- 2.) Attribute (Eigenschaften / Daten)
- 3.) Operationen (Methoden)

Vor den Attribut- oder Methodennamen können Sichtbarkeitssymbole stehen: + für public, - für private und # für protected. Klassenattribute und -methoden werden unterstrichen. Abgeleitete Attribute (automatisch berechnete) werden durch einen vorangestellten Schrägstrich markiert: /abgeleitetesAttr. Abstrakte Methoden werden entweder kursiv geschrieben oder um den Eigenschaftswert {abstract} ergänzt. Vor den Methodennamen können Stereotypen angegeben werden (z.B. «constructor»).

Die Notation für Objekte ist ähnlich der Klassennotation. Statt des Klassennamens wird der unterstrichene Objektname eingesetzt, eventuell gefolgt von einem Doppelpunkt und dem Namen der instanziierten Klasse ("objektName: Klassenname").



Instanzbeziehung

Der Pfeil ist offen und gestrichelt und zeigt vom Objekt zur Klasse (Objekt "ist abhängig von" Klasse, «instance of»).

Der Objektname wird unterstrichen.

Vererbung (Inheritance)

Der Pfeil ist geschlossen und durchgezogen und zeigt von der abgeleiteten Unterklasse (= Subklasse) zur Oberklasse (= Basisklasse = Superklasse).

Die Oberklasse ist eine Generalisierung der Unterklasse und umgekehrt ist die Unterklasse eine Spezialisierung der Oberklasse ("Oberbegriff"-Beziehung = "Ist-ein"-Beziehung).

Abstrakte Klasse

Abstrakte Klassen werden durch den Eigenschaftswert {abstract} oder durch einen *kursiv* gesetzten Klassennamen markiert. Das Aussehen des Vererbungspfeils ändert sich nicht.

Abstrakte Klassen können abstrakte Methoden enthalten, also Methoden deren Funktion noch nicht definiert ist.

Abstrakte Klassen können nicht instanziiert werden, sondern nur als Vererbungsvorlage dienen.

Assoziation, Aggregation und Komposition

Assoziation stellen Beziehungen zwischen Klassen dar. Sie werden in Programmiersprachen meistens durch entsprechende Referenzattribute in den beteiligten Klassen realisiert, bei höherer Multiplizität mit einem Sammlungsobjekt (Collection).

Meistens werden Assoziationen detaillierter als in den gezeigten Abbildungen dargestellt.

An beiden Enden der Verbindungslinie wird meistens die Multiplizität vermerkt, die angibt, mit wie vielen Objekten diese Beziehung zustande kommen kann. Beispiele: '1', '0..1', '0..*', '5..8', '5,8'. Es wird unterschieden zwischen Kardinalität (Anzahl der Elemente) und Multiplizität (Bereich erlaubter Kardinalitäten).

Weiter können an den Enden der Verbindungslinie die für die Beziehung relevanten Rollennamen eingetragen sein.

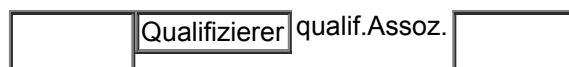
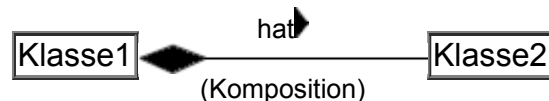
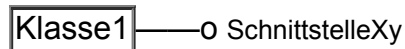
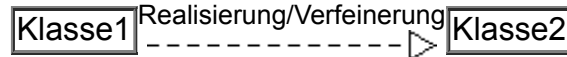
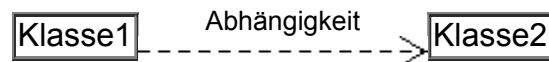
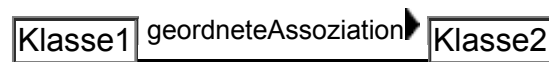
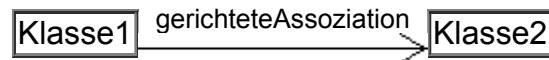
In der Mitte der Verbindungslinie können hinzugefügt werden: «Stereotyp», Beziehungsname, Leserichtung in Form eines dreieckigen Pfeils (▶) und Eigenschaftswerte in eckigen Klammern ([]).

Verfeinerungsbeziehungen («refine», «bind») gibt es zum Beispiel bei der Erzeugung (makroartige Textersetzung) von parametrisierten konkreten Klassen (parameterized Class, bound Element) aus schablonenartigen parametrisierbaren Klassen (Template).

Realisierungsbeziehungen («realize») gibt es zum Beispiel zu Schnittstellen («interface»). Oft wird als Kurzschreibweise die Schnittstellenklasse lediglich durch das "Lolli"-Symbol (Kreis mit Stiel) und dem Schnittstellennamen dargestellt.

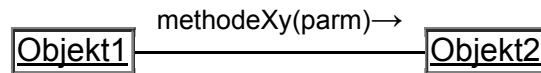
Aggregationen und Kompositionen sind spezielle Assoziationen, die "Teile/Ganzen"-Beziehungen und "Hat-eine"-Beziehungen darstellen. Bei der Aggregation können die "Teile" des "Ganzen" auch einzeln existieren, bei der Komposition nur, wenn auch das "Ganze" existiert (z.B. Rechnungspositionen auf einer Rechnung). Die Verbindungslinie erhält auf der Seite des "Ganzen" eine Raute, die bei der Aggregation ungefüllt und bei der Komposition gefüllt ist. Bei automatischer Codeerzeugung werden Aggregationen und Kompositionen allerdings bei manchen Tools nicht anders als normale Assoziationen behandelt.

Die qualifizierte Assoziation unterteilt die referenzierten Objekte durch qualifizierende Attribute





in Partitionen. Der Qualifizierer ist vergleichbar mit einem Schlüssel (Key) für Dictionaries, assoziative Felder oder Datenbank-Look-up-Tabellen. Der Qualifizierer könnte zum Beispiel eine Personalnummer sein.



Nachrichtenaustausch, Methodenaufruf

Nachrichtenaustausch zwischen Objekten wird durch Methodenaufrufe bewerkstelligt, deren Namen und Parameter zusammen mit einem Pfeil dargestellt werden.

MDA

Definition

MDA (Model Driven Architecture) ist ein Standard der OMG (<http://www.omg.org/mda>) und definiert eine Vorgehensweise beim Softwareentwicklungsprozess unter Verwendung von UML. MDA unterscheidet zwischen PIM (Platform Independent Model), PSM (Platform Specific Model) und Code und bietet eine Grundlage für automatische Code-Generierung.

Begriffe

- MDA:
MDA (Model Driven Architecture) definiert eine Vorgehensweise beim Softwareentwicklungsprozess (<http://www.omg.org/mda>).
- UML 2.0:
MDA basiert auf UML in der Version 2.0 (Unified Modeling Language) ([techdocs/uml.htm](http://www.omg.org/uml), <http://www.omg.org/uml>).
- MOF, XMI:
Die UML-Metadaten der OO-Modelle werden in MOF-Repositories (Meta-Object Facility) gespeichert und können per XMI (XML Metadata Interchange) mit anderen UML- oder MDA-Tools ausgetauscht werden (<http://www.omg.org/technology/documents/formal/mof.htm>, [.../xmi.htm](http://www.omg.org/technology/documents/formal/xmi.htm)).
- PIM, PSM:
Die Modellierung der fachlichen Geschäftsprozesse (PIM, Platform Independent Model) ist von der Modellierung der technischen Realisierung (PSM, Platform Specific Model) getrennt.

- Austauschbare technische Plattform:
Die späte Wahl bzw. die spätere Umentscheidung der verwendeten technischen Plattform (z.B. Java EE, .NET) ist möglich.
- Patterns und Wiederverwendung:
Bewährte Design Patterns, Best Practices und wiederverwendbare Komponenten werden unterstützt.
- Vorgehensmodelle:
MDA ist mit etablierten Entwicklungsprozessen und Vorgehensmodellen vereinbar (z.B. RUP, Agile Modeling, XP).
- Automatisierung:
Sowohl die Transformation des PIMs in das PSM (z.B. mit Java EE Patterns) als auch die anschließende Generierung von Sourcecode (z.B. Java) kann eventuell (zumindest weitgehend) automatisiert werden (z.B. für Webanwendungen).

Mögliche Anforderungen an UML-/MDA-Tools zur automatischen Codegenerierung

- "Echtes" MDA oder nur MDD-Variante?
Aufteilung in PIM (Platform Independent Model), PSM (Platform Specific Model) und Code?
- OCL (Object Constraint Language)? Oder Alternative?
- Code-relevante Ausgangsbasis nur UML-Klassendiagramme oder auch andere UML-Diagrammtypen?
- Welche automatisch erzeugten Ergebnis-UML-Diagrammtypen?
- Welche Design Patterns? GoF und Sun? Können Patterns verändert oder eigene Patterns verwendet werden? Nur gegen Aufpreis?
- Forward, Reverse, Round-trip?
- Falls kein Round-trip: Ist Einbetten von eigenem Code möglich (z.B. in "geschützten Bereichen"), der auch nach Rebuild erhalten bleibt?
- Konform zu XMI (XML Metadata Interchange), MOF (Meta-Object Facility) und JMI (Java Metadata Interface)?
- Mit welchen anderen UML-/MDA-/RAD-Tools ist welcher Datenaustausch möglich (z.B. Rational Rose XDE, Borland Together)?
- Ziel-Programmiersprachen/-Plattformen (z.B. Java-Standalone, JSP+Tomcat, EJB+JBoss, .NET)?
- GUI-Builder?
- Können Stand-alone-Applikationen entwickelt werden?
Mit Swing und SWT (Standard Widget Toolkit)?
- Können Webapplikationen entwickelt werden? Mit Struts? Mit JSF (JavaServer Faces)?
Komfortable Erstellung des GUIs für Webapplikation (inklusive Tree-Views und horizontale Menüleisten mit Pulldown-Menüs)?
Für die Webbrowser Microsoft Internet Explorer, Netscape, Mozilla, Opera und Konqueror?
Auch für mobile Endgeräte (WAP / WML / MIDP)?
- Deployment per Mausklick?
Automatisches Deployment für Tomcat, JBoss, WebLogic und WebSphere?

Deployment für welche weiteren Plattformen / Frameworks?

- Können Portlets erstellt werden? Für welche Portal-Server? Auch nach "JSR 168 Portlet Specification"? Auch nach WSRP-Standard (Web Services for Remote Portlets)?
- Erzeugung eines eigenen und Anbindung an einen fremden SOAP Web Service per Mausklick?
- Automatische Erzeugung eines GUIs für fremden Web Service anhand der WSDL-Datei per Mausklick?
- Per Mausklick: JMS, CORBA, EAI (z.B. mit SAP R/3® BAPI)?
- Unterstützung für Single-Sign-on, LDAP und SSL?
- Anbindung an welche SQL-Datenbanken (MySQL, PostgreSQL, MaxDB, Oracle, IBM DB2, MS SQL Server)?
- Komfortables Debugging? Auch Remote?
- Testsoftware per Mausklick? JUnit?
- Welche Phasen des kompletten Entwicklungszyklusses werden abgedeckt?
Welche Tools werden für die fehlenden Phasen empfohlen?
Komfortable Einbindung in CVS und Subversion?
Ist Einbettung in Vorgehensmodelle wie RUP möglich?

Links auf weiterführende Informationen

■ Websites

- UML bei OMG: <http://www.omg.org/uml>
- UML bei Jeckle: <http://www.jeckle.de/unified.htm>, <http://www.jeckle.de/files/telelogic2003.pdf>,
<http://www.jeckle.de/uml-glasklar/AgilityDays2003.pdf>
- UML bei Gulp: <http://www.gulp.de/kb/it/projekt/projekterfolg1.html>
- UML-Tutorial, Universität Magdeburg: <http://www-ivs.cs.uni-magdeburg.de/~dumke/UML>
- UML, Universität Oldenburg: <http://elvis.offis.uni-oldenburg.de/Zwischenbericht-A/node24.html>
- UML bei Rätzmann: http://www.dfpug.de/konf/konf_1999/gruppe02_oop/d_uml/d_uml.htm
- Sun Java Center, J2EE Patterns: <http://java.sun.com/developer/technicalArticles/J2EE/patterns>

- Architecture and Architecture Modeling Techniques (UML, MDA, EUP): <http://www.agiledata.org/essays/enterpriseArchitectureTechniques.html>
- MDA, Code Generation Network: <http://www.codegeneration.net/tiki-index.php?page=MDA>
- openMDA: <http://www.openmda.de/aboutmda.htm>
- MDA from a Developer's Perspective, TheServerSide.com: <http://www.theserverside.com/resources/article.jsp?!=MDA>
- Model Driven Architecture, software-kompetenz.de: <http://www.software-kompetenz.de/?5348>
- Model Driven Architecture, Szallies: <http://www.wohnsklo.de/szallies/mda>
- **Zeitschriftenartikel**
- Model Driven Architecture, Grundlegende Konzepte und Einordnung der Model Driven Architecture (MDA), Roßbach/Stahl/Neuhaus, [Javamagazin 2003.09, Seite 22](#)
- **Bücher**
- Jeckle/Rupp/Hahn/Zengler/Queins, UML 2 glasklar, 2003-11, 3446225757: [Amazon.de](#)
- Chonoles/Schardt, UML 2 für Dummies, 2003-12, 3826630912: [Amazon.de](#)
- Oestereich, Objektorientierte Softwareentwicklung (mit UML 2), 2004, 3486272667: <http://www.oose.de/publikationen.htm>, [Javamagazin](#), [Amazon.de](#)
- Born/Holz/Kath, Softwareentwicklung mit UML 2, 2003-12, 3827320860: [Amazon.de](#)
- Rumbaugh/Jacobson/Booch, The Unified Modeling Language Reference Manual, 1998, 020130998X: [Amazon.de](#)
- Andresen, Komponentenbasierte Softwareentwicklung mit MDA, UML und XML, 2003, 3446222820: [Amazon.de](#)
- Warmer/Kleppe/Bast, MDA Explained, 2003, 032119442X: [Amazon.de](#); (berücksichtigt OptimaIJ)
- Frankel, Model Driven Architecture, 2003, 0471319201: [Amazon.de](#)
- Hubert, Convergent Architecture, Building Model-Driven J2EE Systems with UML, 2001, 0471105600: convergentarchitecture.com, [Amazon.de](#); (berücksichtigt ArcStyler)
- **UML-Tools**
- Siehe auch Javamagazin 2003.03, Seite 44, "Ein Überblick über die am Markt befindlichen UML-Produkte"
- UML-Tools-Listen: <http://www.oose.de/service/uml-werkzeuge.html>, http://www.objectsbydesign.com/tools/umltools_byCompany.html
- Uebersicht zu UML-MDA-Tools: <http://www.ecotronics.ch/kleiner/umltools.htm>
- yEd, yWorks: http://www.yworks.com/en/products_yed_about.html
- astah (früher JUDE), Change Vision: <http://astah.change-vision.com/en/product/astah-community.html>
- ArgoUML, Tigris.org: <http://argouml.tigris.org>
- Poseidon, Gentleware: <http://www.gentleware.com>

- UML-Plug-in für Eclipse: <http://eclipseuml.com>
- Enterprise Architect, Sparx Systems: <http://www.sparxsystems.com.au/ea.htm>
- **UML-Entwicklungsumgebungen**
- Rational Rose XDE Developer, IBM: <http://www.rational.com/products/xde>
- Rational Rapid Developer (RRD), IBM: <http://www.rational.com/products/rapiddeveloper>
- Together, Borland: <http://www.borland.de/together>, <http://www.borland.de/together/solo>, <http://www.togethersoft.com/products>
- **MDA-Entwicklungsumgebungen**
- Ameos, Aonix, Karlsruhe/München/Korschenbroich: <http://www.aonix.de>
- AndromDA: <http://www.andromda.org>, <http://sourceforge.net/projects/andromda>, <http://genome.tugraz.at/Theses/Truskaller2003.pdf>
- ArcStyler, Interactive Objects Software, Freiburg: <http://www.arcstyler.com>, http://www.arcstyler.com/products/arcstyler_overview.jsp, http://www.io-software.com/news/pr_mai_200503_d.jsp
- iQgen, innoQ, Ratingen: <http://www.innoq.com/iqgen>
- MDE, Metanology: <http://www.metanology.com>
- OptimalJ, Compuware: <http://www.compuware.de/products/optimalj>, <http://www.compuware.no/optimalJ/ButlerGroupReport.pdf>
- smartGENERATOR, BITPlan, Meerbusch: <http://www.bitplan.de/xmlweb/index.php?topic=products/smartgenerator>
- Software through Pictures (StP), Aonix: http://www.aonix.de/stp_acd.html, <http://www.jfs2003.de/abstracts.html#A4> / [PDF](#)
- XCoder: <http://sourceforge.net/projects/xcoder>, <http://www.liantis.com/Leistungen>

Weitere Themen: [andere TechDocs](#) | [Vorgehensmodelle](#)

© 2001-2007 Torsten Horn, Aachen