Universität Rostock Traditio et Innovatio

Professur für Geodäsie und Geoinformatik
Prof. Dr.-Ing. Ralf Bill
M.Sc. Markus Berger

Tutorial - Vorlesung: Modellierung in UML und ER

Modellierung in UML und ER

In dieser Lerneinheit werden die wesentlichen Grundzüge der Modellierung vorgestellt und am Beispiel der Unified Modeling Language (UML) und des Entity-Relationship-Modells (ER) erläutert. In einem separaten Übungstutorial kann das Gelernte dann praktisch vertieft werden.	 Bildquelle: [1]

Einführung

Die Probleme der Geoinformatik haben stets einen Bezug zur realen Welt. Wir vermessen wirklich existierende Objekte, wir arbeiten auf Daten, welche eben diese repräsentieren und wir planen Strukturen, die in der echten Welt aufgebaut werden sollen. Dieser Fakt stellt uns vor ein Problem: Die Realität ist komplex. Umso weiter man sich in ein Problem hineinarbeitet, umso mehr ungeahnte Zusammenhänge tun sich auf. Nehmen wir das Beispiel eines Baumkatasters: Die grundlegende Idee ist es, ein Verzeichnis über vorhandene Bäume in einem Gebiet zu führen. Aber welche Informationen wollen wir aufnehmen? Dass es unmöglich ist jeden Baum in seiner Gesamtheit zu beschreiben, ist schnell erkenntlich. Der Zustand jedes einzelnen Blattes ist nicht nur uninteressant, sondern auch nach dem Biss eines Käfers bereits hinfällig.

Stattdessen versuchen wir die Realität und ihre Systeme auf ein für uns verständliches Maß hinunter zu brechen. Unser Gehirn tut dies ganz natürlich in jedem Moment unserer Wahrnehmung. Definiert man ein System als eine Gesamtheit von miteinander verbundenen Elementen, dann ist es offensichtlich, dass wir stets nur mit Wissen über eine Teilmenge dieser Elemente arbeiten. Diese Teilmenge wird weiter dadurch eingeschränkt, dass sie für die (Geo-)Informatik informationstechnisch verarbeitbar sein muss.

Modell

Um herauszuarbeiten, wie genau diese Teilmenge aussieht, und um sie zu verstehen, nutzen wir Modelle. Ein Modell ist eine vereinfachte Abstraktion eines realen Systems. Wird eine Modellierung vorgenommen, verfolgen wir üblicherweise die folgenden Ziele:

1. Das Modell soll uns helfen ein existierendes oder geplantes System zu **visualisieren**.
2. Das Modell soll die Struktur und das Verhalten eines Systems **spezifizieren**.
3. Das Modell soll während der Implementierung eines Systems **als Referenz dienen**.
4. Das Modell soll unsere Implementierungsentscheidungen **dokumentieren**.

[1] Bildquelle: https://de.wikipedia.org/wiki/Unified_Modeling_Language#/media/File:UML_logo.svg

Damit ein Modell alle diese Anforderungen erfüllt, sind drei Eigenschaften Pflicht. Egal wie abstrahiert, das Modell muss stets eine **Reproduktion** eines bestehenden oder geplanten Systems sein. Je nach Abstraktionslevel geschieht eine **Reduktion** auf relevante Aspekte des Originalsystems. Dabei muss stets die **Praktikabilität** im Bezug zur derzeitigen Aufgabe im Blick gehalten werden.

Anhand dieser Eigenschaften sollte klarwerden, dass es für kein in der Realität existierendes System nur ein Modell geben kann. Es muss stets auf die gewünschte Präzision und den gewünschten Kontext geachtet werden. Für nicht-triviale Systeme werden daher meistens Sammlungen an Modellen benötigt, welche ihren Fokus auf verschiedene Sachverhalte legen. Die Wahl dieser Modelle kann einen tiefgreifenden Einfluss auf die spätere Arbeit mit einem System haben.

Auf den Punkt bringen lässt sich die Arbeit mit Modellen durch zwei vielzitierte Weisheiten:

„So simpel wie möglich, so komplex wie nötig."

„Alle Modelle sind falsch, aber einige sind nützlich."

Um Modelle nutzbar zu machen, müssen sie auf irgendeine Art ausgedrückt werden. Meist passiert dies grafisch, teilweise auch textuell oder mittels physischer Objekte.

Einige Beispiele für *Modelle,* auf die jeder schon mal gestoßen ist:

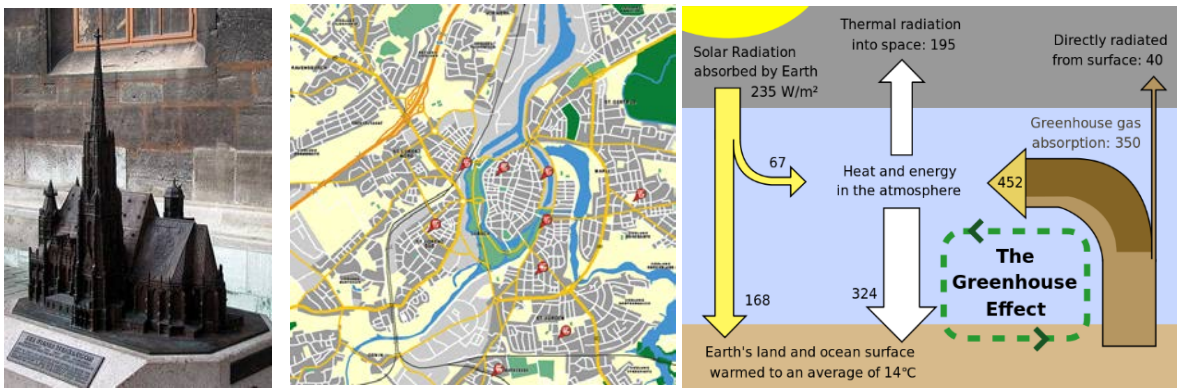

Abbildung 1: Von links nach rechts: Ein maßstabsgetreues Modell eines echten Gebäudes, eine Kartenvisualisierung einer Siedlung, ein Flussdiagramm zum Treibhauseffekt.

Sie alle zeigen eine vereinfachte Repräsentation einer realen Struktur oder eines realen Ablaufs. Außerdem dienen sie alle der Visualisierung. Sie sollen schwer überblickbare Sachverhalte einfangen und einem Betrachter verständlich machen.

Wir wollen jedoch Modelle für Probleme der Geoinformatik einsetzen. Und als solche sind dies informationstechnische Probleme. Wie stellen wir einen Sachverhalt in einer Datenbank dar? Wie ordnen wir Messergebnisse ein, und welche Werte sind für unsere Aufgabe interessant? Wie hängen die Schritte eines Planungs- und Implementierungsprozesses voneinander ab?

Modellierungssprachen

Um solche Fragen zu behandeln, werden sowohl in Wirtschaft als auch Wissenschaft seit Jahren spezielle Modellierungssprachen eingesetzt. Diese standardisieren den Modellierungsvorgang, und bringen viele Vorteile mit sich. Jeder, der mit der entsprechenden Sprache vertraut ist, kann sofort Schlüsse aus dem Modell ziehen und selbst Änderungen vornehmen. Ist das Modell maschinenlesbar, sorgt die Sprache für Kompatibilität zwischen verschiedenen Tools. Und auch wenn man ohne Tools oder andere Menschen modelliert, erlaubt sie sich auf die Problemlösung zu konzentrieren, nicht auf den Neuentwurf einer passenden Repräsentationsform.

Aufgrund der Nähe der Geoinformatik zur grundlegenden Informatik, wird meistens auf die zwei dort populärsten Modellierungssprachen zurückgegriffen:

Unified Modelling Language (UML) **Entity-Relationship Modell (ER-Modell)**

Das ER-Modell stammt aus dem Bereich der Datenbanksysteme, und beschreibt Datenstrukturen so wie sie später in den Tabellen einer relationalen Datenbank wiedergespiegelt werden sollen. UML beschreibt nicht nur Datenstrukturen, sondern auch Prozesse und Systemteilnehmer. Die Sprache ist dabei mit einem Fokus auf objektorientierte Datenhaltung entworfen, wie sie oft in der Programmierung benutzt wird. Nutzer werden daher oft auf Konzepte wie Klassen, Objekte und Vererbung treffen. UML ist eine sehr flexible Modellierungssprache, und bietet die Möglichkeit für jedes ER-Modell ein UML-Äquivalent zu formulieren. Gleichzeitig ist sie weitaus komplexer, kann das selbe Modell einem Leser damit auch schwerer verständlich machen. Hier sollte immer genau geprüft werden, welche Sprache sich für die Aufgabe und das Zielpublikum am besten eignet.

Auf diese Art und Weise tragen wir nun alle Attribute und Methoden aus der Vorlesung ein. (Jeweils den letzten im Dokument auffindbaren Stand.). Dies sieht im Ergebnis wie folgt aus:

Modellierungssprachen

Grundlagen

ER-Modelle können zwar verbal ausgedrückt werden, sind jedoch in praktischen Anwendungen fast immer grafisch repräsentiert. Man spricht bei diesen Repräsentationen von ER-Diagrammen.

Für Datenbanken mit hoher Komplexität ist es unabdingbar sich vor der Implementierung Gedanken über die Struktur zu machen. Später umfassende Änderungen vorzunehmen ist nur sehr schwer möglich, da oft viele Abhängigkeiten und Referenzen vorliegen. Mittels ER-Modell kann man schnell prüfen ob die gewünschte Abstraktion informationstechnisch Sinn ergibt.

Schauen wir uns zunächst an einem Beispiel an von was für einer Abstraktion wir hier reden. Datenbanken sind im Kern Sammlungen von Tabellen, deren Daten sich untereinander referenzieren können. Solche Tabellen werden in der Welt der Datenbanken Relationen genannt. Ab hier werden wir der Klarheit halber ebenso diesen Begriff verwenden. Nehmen wir also folgende zwei Relationen wie sie in einem Baumkataster einer Gemeinde stehen könnte:

Tabelle 1: Baum-Relation

Baum-ID	Art	Höhe	Gemeinde-ID	Koordinaten
001	Birke	4	1302	Punkt(20,10)
002	Fichte	6	1303	Punkt(0,100)
003	Eiche	6	1302	Punkt(20,30)

Tabelle 2: Gemeinde-Relation

Gemeinde-ID	Gemeindename	Einwohnerzahl
1301	Buchholz	3500
1302	Laubitz	1200
1303	Fichtenwerder	8300

Worum es bei der Modellierung ausdrücklich nicht geht, sind die einzelnen Datensätze/Zeilen in diesen Relationen. Einzelne Bäume (konkrete Objekte unserer Wirklichkeit) müssen nicht modelliert werden. Stattdessen modellieren wir was wir über die Bäume abspeichern, und wie. Über Detaillevel, das „was" haben wir schon in der Einleitung gesprochen, d.h. es ist für ein Baumkataster unnötig die DNA-Sequenz des Baumes zu speichern. Für ER-Modelle ist allerdings das „wie" noch interessanter. Gemeindenamen und Einwohnerzahl könnten wir direkt in der Baum-Relation speichern. Dies macht informationstechnisch jedoch wenig Sinn, da so bei einer einfachen Änderung der Einwohnerzahl diese auch in allen Baumdatensätzen geändert werden müsste. Daher speichern wir in der Relation nur eine eindeutige Referenz zu jedem Gemeinde-Datensatz. So können beide Datensätze unabhängig voneinander bearbeitet werden. Das ER-Modell soll uns helfen, solche Abhängigkeiten frühzeitig zu planen, und entsprechend aufzubauen.

Entitäten und Relationen

Wie sieht nun so ein Modell aus? Wie der Name sagt beginnen wir mit *Entitäten*. Ein einzelner Datensatz (eine Zeile) ist eine *Entität*. Eine ganze Relation wird repräsentiert durch einen *Entitätstyp.* Ein *Entitätstyp* wird in ER-Diagrammen durch ein Rechteck mit dem Namen der Relation dargestellt:

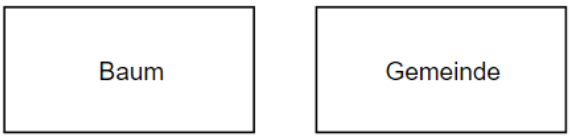

Abbildung 2: Entitätstypen als Rechtecke

Wie werden nun die Abhängigkeiten dargestellt? Dafür ist zuerst einmal der zweite Teil des Namens zuständig: *Relationships*, zu Deutsch *Beziehungen*. Eine *Beziehung* besteht immer zwischen zwei *Entitäten*. In unserem obigen Beispiel **befindet sich** jeder Baum **in** einer Gemeinde. Die *Beziehung* zwischen einem Baum und einer Gemeinde könnte daher z.B. „**befindet sich in**" heißen. Auch hier arbeiten wir jedoch nicht auf Ebene der *Beziehungen*, sondern mit *Beziehungstypen*. Diese fassen die *Beziehungen* zwischen den einzelnen *Entitäten auf Entitätstypen*-Ebene zusammen, und nutzen dafür sogenannte *Kardinalitäten*. Diese beschreiben wie viele *Entitäten* auf jeder Seite der *Beziehung* beteiligt sein können. So befindet sich ein Baum immer in genau einer Gemeinde, die Gemeinde selbst kann jedoch gar keine (0) bis beliebig viele Bäume (nachfolgend mit „N" bezeichnet) enthalten.

Dies wird in ER-Diagrammen über beschriftete Linien dargestellt:

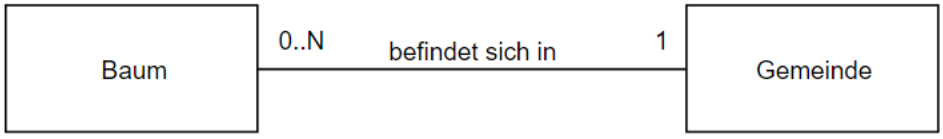

Abbildung 3: Beziehungstypen als beschriftete Linien

Primärschlüssel

Es fehlt nun nur noch ein Bauteil um mehrere *Relationen* und ihre Beziehungen darzustellen: Die Spalten der *Relationen*. Man spricht hier von *Attributen*. Diese sind nicht nur von inhaltlicher Wichtigkeit, sondern sind auch bedeutend bei der Implementierung der Beziehungen. In unserer Beispieltabelle gab es z.B. das *Attribut* „Gemeinde-ID“. Diese hat eine Beziehung von jedem Baum zu genau einer Gemeinde hergestellt. Ein *Attribut* oder eine Sammlung von *Attributen,* welche einen Datensatz in einer *Relation* eindeutig identifizieren, nennt man *Primärschlüssel*.

In ER-Diagrammen werden Attribute durch an Relationen gebundene Ellipsen, und Primärschlüssel durch doppelte Ellipsen dargestellt.

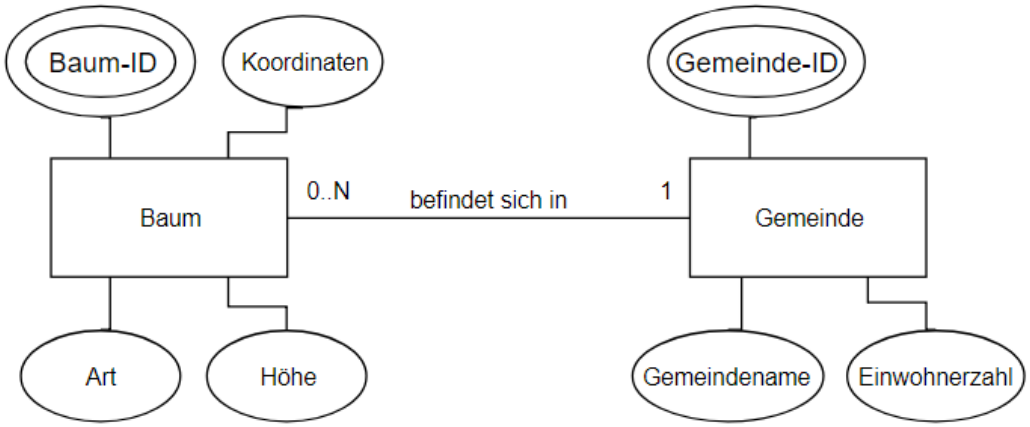

Abbildung 4: Attribute und Primärschlüssel als Ellipsen

Aufmerksamen Lesern ist vielleicht aufgefallen, dass das Gemeinde-ID *Attribut* nicht mit dem Baum-Entitätstyp verbunden ist, obwohl es in dessen *Relation* vorkommt. Dies liegt daran, dass dieses *Attribut* nur als sogenannter *Fremdschlüssel* in der *Relation* enthalten ist, also als Referenz auf den *Primärschlüssel* einer anderen *Relation*. Eine *Beziehung* impliziert mindestens einen solchen *Fremdschlüssel*.

Aus einem solchen *ER-Diagramm* kann ein Datenbankentwickler eindeutig ablesen, welche *Relationen* und Schlüssel benötigt werden. Ein Problem ist jedoch in unserem Diagramm noch versteckt: Wir haben der Gemeinde eine „0..N“ *Kardinalität* zu den Bäumen zugestanden. Diese ist in unseren anfänglichen Tabellen nicht repräsentiert. Um sie zu erreichen müsste für jede Gemeinde eine Liste von Baum-IDs geführt werden. Da diese in einer wohlkonstruierten Datenbank jedoch nicht in einem Feld zusammengeführt werden können, wird hier typischerweise eine neue *Relation* kreiert, welche nur die beiden *Primärschlüssel* der *Relationen* als *Attribute* enthält. So wird jede einzelne *Beziehung* als Paar abgespeichert. *Relationen* können also nicht nur aus *Entitätstypen* entspringen, sondern auch aus *Beziehungstypen*.

Für weitere Details zur Benutzung von *ER-Modellen* mit Datenbanken, siehe Heuer/Saake/Sattler (2007).

Unified Modeling Language (UML)

Grundlagen

Die *UML* bietet weitaus mehr Möglichkeiten als *ER-Modelle*. Sie enthält verschiedene Arten von Diagrammen für verschiedene Verwendungszwecke. Ursprünglich stammt sie aus dem Feld der Softwareentwicklung, hat sich jedoch inzwischen in Wirtschaft, Politik und Forschung verbreitet.

Genau wie *ER-Modelle* wird sie meistens in Form von Diagrammen visualisiert, bietet jedoch auch textuelle Repräsentationen. Für fortgeschrittene Nutzungsformen werden Konzepte wie Metamodellierung und Domänenprofile angeboten. Diese sind jedoch außerordentlich komplex, und in vielen Fällen überflüssig. Wir legen hier den Fokus auf eine Auswahl der grundlegenden UML-Diagrammtypen.

Wie schon angedeutet basiert *UML* auf einer objektorientierten Denkweise. Diese ist der relationalen Denkweise der *ER-Modelle* nicht komplett unähnlich, bietet jedoch einige neue Möglichkeiten. Im Zentrum der Objektorientiertheit steht – natürlich – das *Objekt*. Ein *Objekt* entspricht einer *Entität* im *ER-Modell*. Es enthält Attributfelder, in welchen die Daten stehen. Welche Attributfelder ein Objekt besitzt hängt von seiner *Klasse* ab. *Klassen* sind das Äquivalent von *Entitätstypen*. *Beziehungen*, *Beziehungstypen* und *Kardinalitäten* existieren hier genau wie in *ER-Modellen*, und nennen sich in ihrer grundlegenden Variante *Assoziationen*. Schlüsselattribute müssen jedoch nicht mehr speziell markiert werden, eine Erklärung dazu folgt für Interessierte in der Box.

Primär- und Fremdschlüssel werden mit dem objektorientierten Paradigma nicht mehr benötigt. Stattdessen werden direkte Verweise auf andere Objekte genutzt. Die Datenspeicherung wird nicht über Tabellen gehandhabt, sondern man spricht stets einzelne Objekte an. Will man alle Objekte einer Klasse verändern, muss man vorher eine Liste mit Referenzen angelegt haben. Ist ein Objekt referenziert, verhält es sich genauso wie jedes andere Attribute, bloß ist ihre Klasse nicht „Textfeld“ oder „Ganzzahl“, sondern „Baum“ oder „Liste(Gemeinde)“.

Bisher würde unser *UML-Diagramm* als *Klassendiagramm* fast genauso aussehen wie ein *ER-Diagramm*. Baum und Gemeinde sind *Klassen*, einzelne Bäume und Gemeinden *Objekte. Beziehungen* bleiben gleich, nur der *Primärschlüssel* fällt weg. *Attribute* werden jedoch anders notiert. Sie werden mit dem *Objekt* in das Rechteck geschrieben, und der Datentyp muss stets spezifiziert werden. Dies ist nötig, da diese weniger selbsterklärend sind als in *Relationen*. Im objektorientierten Paradigma gibt es nicht nur Text, Daten und Nummernfelder, sondern auch andere *Klassen* können als Attributtyp genutzt werden.

Beachte: An dieser Stelle nicht unbedingt relevant, aber durchaus interessant: Dies bedeutet, dass auch grundlegende Attribute wie Zahlen oder Textfeldern intern auf Klassen basieren. Diese Klassen werden als gegeben erachtet, und müssen nicht jedes Mal im Modell eingetragen werden.

Unser finales *ER-Diagramm* würde übersetzt als *UML-Klassendiagramm* also so aussehen:

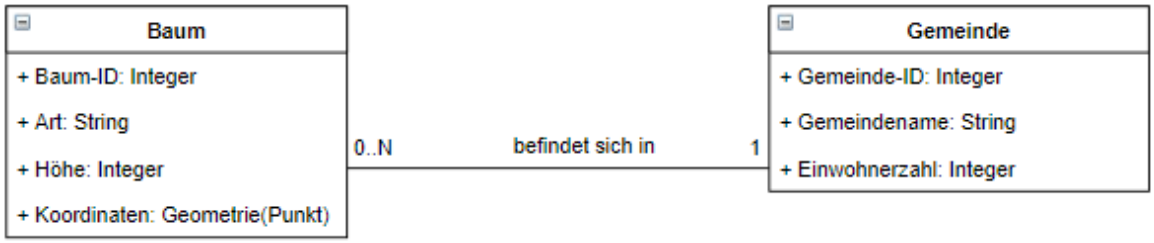

Abbildung 5: Beispiel als UML-Klassendiagramm

Dynamik

Wir nutzen *UML-Diagramme* jedoch nicht wegen der etwas effizienteren Notation. Es gibt tatsächlich fundamentale Unterschiede zwischen *ER*- und *UML-Modellierung*.

Im Kern dieser Unterschiede liegt die Dynamik. *UML-Objekte* sind nicht nur statische Datenspeicher, sondern können Vorgänge ausführen. Man spricht hier von *Methoden*, welche genau wie *Attribute* an *Objekte* angebunden sind. In jeder dieser *Methoden* sind Abfolgen von Schritten gespeichert. Meist geht es hier um das Verändern von Daten. Ein Baum-*Objekt* könnte z.B. eine *Methode* „Umpflanzen“ haben, welche das „Gemeinde“ und das „Koordinaten“-*Attribut* des Baumes verändert. Dafür werden natürlich die neue Gemeinde und die neuen Koordinaten benötigt. Diese können daher beim Aufrufen von Methoden „übergeben“ werden, als sogenannte *Parameter*. Andere *Methoden* geben einfach nur Informationen über das *Objekt* an den Aufrufenden zurück, ohne *Parameter* zu benötigen. So könnte eine *Methode* eines Baums „gebeRealhöhe“ heißen, welche die Baumhöhe auf die Höhe über NN addiert und als Zahl zurückgibt. Genau wie die *Attribute* haben *Methoden* damit einen Datentyp/eine *Klasse*, welche/r aussagt was für Daten sie zurückgeben.

Unser Beispielmodell könnte mit Dynamik z.B. so aussehen:

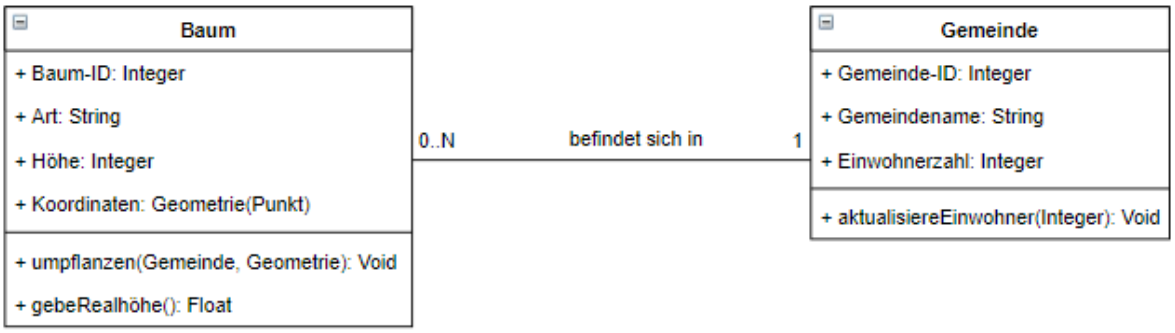

Abbildung 6: Beispiel mit Methoden (Hinweis: Der Typ „Void“ wird benutzt wenn eine Methode keine Daten an den Aufrufenden zurückgibt)

Der zweite große Unterscheidungspunkt liegt in den fortgeschrittenen Konzepten der Objektorientierung, insbesondere der sogenannten *Vererbung*. Diese werden in der Sektion über Klassendiagramme genauer erklärt.

UML-Diagrammtypen

Am Anfang des Kapitels haben wir erwähnt, dass UML mehrere Arten von Diagrammen anbietet. Dies liegt vor allem daran, dass dynamische Vorgänge modelliert werden können. Es ist nicht nur möglich, aufzulisten was für Methoden vorliegen, sondern auch wie diese voneinander abhängen, an welchem Punkt welches Objekt angesprochen wird, oder wer diese Methoden in welchem Anwendungsfall aufruft. In der aktuellen Version von UML (2.5.1) sind 14 Diagrammtypen enthalten. Sie unterteilen sich in Struktur- und Verhaltensdiagramme, je nachdem ob sie sich mit statischen oder dynamischen Vorgängen befassen. Verhaltensdiagramme enthalten auch Modelle für die Interaktion des Benutzers mit Daten und Methoden.

Das von uns in Abbildung 6 erstellte Diagramm ist ein sogenanntes Klassendiagramm, der am häufigsten verwendete Diagrammtyp. Es kann strukturelle Aspekte eines Systems schnell und effizient visualisieren, und dient oft als Grundlage für Planung und Implementierung. Es handelt sich um ein Strukturdiagramm. Im Folgenden werden wir erst genauer auf das Klassendiagramm eingehen, da einige wichtige Aspekte in unserem Beispiel noch nicht gezeigt wurden. Danach werden wir uns ein Beispiel für ein Verhaltensdiagramm ansehen.

Die folgende Tabelle listet alle derzeit definierten Typen auf:

Strukturdiagramme	**Verhaltensdiagramme**
Klassendiagramm	Aktivitätsdiagramm
Komponentendiagramm	Anwendungsfalldiagramm
Kompositionsstrukturdiagramm	Interaktionsübersichtsdiagramm
Verteilungsdiagramm	Kommunikationsdiagramm
Objektdiagramm	Sequenzdiagramm
Paketdiagramm	Zeitverlaufsdiagramm
Profildiagramm	Zustandsdiagramm

Klassendiagramm

Das vorher beschriebene Diagramm ist schon ein valides *Klassendiagramm*. Im Gegensatz zu *ER-Diagrammen* bieten *Klassendiagramme* jedoch noch weitere Möglichkeiten. Die wichtigsten davon sind die Konzepte der *Generalisierung* und der *Aggregation*.

Bei der *Generalisierung* verbinden wir *Klassen* miteinander, wobei eine der *Klassen* spezieller und die andere genereller ist. In anderen Worten: Eine *Klasse* abstrahiert die andere. Wir könnten also eine *Klasse* „Pflanze“ haben, welche die *Klasse* „Baum“ generalisiert. Ein Baum ist eine spezielle Art von Pflanze. Solche Beziehungen sind sehr häufig in der objektorientierten Modellierung, und werden durch einen weißen Pfeil gekennzeichnet. Der Effekt solcher Konstrukte ist, dass die speziellere *Klasse* alle *Attribute* und *Methoden* der generelleren *Klasse* „erbt“, jedoch zusätzlich ihre eigenen Merkmale hinzufügt. So weiß ein Nutzer, dass er an alle Pflanzen die „umpflanzen“-Anfrage stellen kann, egal was für eine Pflanze er vor sich hat. Ein weiterer Effekt ist, dass man sich in der „Gemeinde“-*Klasse* eine Menge Arbeit spart. Man muss nicht für jeden neuen Typ von Pflanze ein neues *Attribut* erstellen, stattdessen ist der Datensatz später um weitere Typen erweiterbar, ohne irgendetwas an der „Gemeinde“-*Klasse* zu ändern.

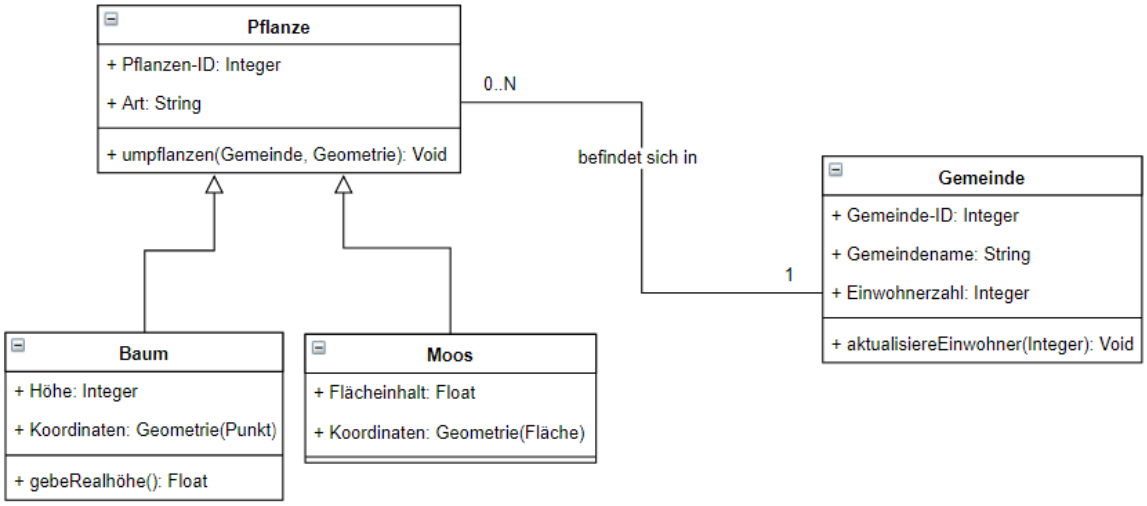

Abbildung 7: Klassendiagramm mit Generalisierung (Weiße Pfeile sind in Pfeilrichtung zu lesen als "spezialisiert")

UML-Beziehungen

Es sollte beim Modellieren immer nach Möglichkeiten zur *Generalisierung* Ausschau gehalten werden, da diese einen der Grundpfeiler der Modellierung mit sich bringen: *Abstraktion*. Diese Zusammenhänge zu finden ist eines der Hauptziele beim Erstellen von *Klassendiagrammen.*

Ein weiteres Konzept ist die *Aggregation*. Hier handelt es sich um eine spezielle Form der *Assoziation*, der Beziehung zwischen zwei Objekten. Während eine *Assoziation* im Grunde nur dafür gedacht ist, dass zwei *Objekte* miteinander kommunizieren und sich referenzieren können, sagt eine *Aggregation* aus, dass eine *Klasse* ein „Container“ für die andere *Klasse* ist. Letztere ist also Teil eines Ganzen. Sie kann sogar Teil mehrerer Container sein, d.h. in einer *Aggregation* ist das Teil nicht von seinem Ganzen abhängig.

Eine *Komposition* ist wiederum eine strengere *Aggregation*, in der das Teil von seinem Ganzen abhängig ist und nur Teil von maximal einem Ganzen sein kann. In einer *Aggregation* sind die Teile vom Ganzen separierbar, in einer *Komposition* nicht. Diese Definition ist nicht ganz intuitiv, und selbst in gut kurierten Quellen oft falsch erklärt und angewendet. Wir versuchen die Idee daher an unserem Beispiel zu verdeutlichen: Nehmen wir an, dass die Verwaltungseinheit unter der Gemeinde „Bezirk“ genannt wird. Außerdem befinden sich in einer Gemeinde meist mehrere Siedlungen (Dörfer, Städte, etc.), welche unter Umständen auch über eine Grenze hinweg in zwei Gemeinden liegen können. Der Bezirk ist hier also eine bürokratische Einordnung, die Siedlung ein real existierendes Konstrukt. Eine dieser Beziehungen ist eine *Aggregation*, die andere eine *Komposition.* Wieso?

Zuerst einmal sind beide *Aggregationen*, da sie Inhalt der Gemeinde sind. Die Gemeinde hat nicht nur eine referentielle Beziehung zu ihnen, sondern Teil ihres Zwecks ist es Bezirke und Siedlungen unter sich zu sammeln. Um nun herauszufinden, bei welcher Beziehung es sich um eine *Komposition* handelt, müssen wir uns folgende Frage stellen: Was passiert, wenn die Gemeinde auf einmal verschwinden würde – z.B. im Rahmen einer Gebietsreform. Die Siedlungen würden natürlich weiter bestehen, da sie im Laufe der Reform hoffentlich nicht abgerissen werden. Hier liegt also eine *Aggregation* vor. Die Bezirke in der Gemeinde würden jedoch verschwinden, da sie nur im Zusammenhang mit dieser Sinn ergeben. Die Gemeinde ist also eine *Komposition* von Bezirken.

Hinweis: Hierbei wird oft ein Fehler gemacht. Nur weil die Teilobjekte bei einer Komposition verloren gehen, wenn ihr Container zerstört wird, muss dies nicht bedeuten, dass sie zwingend einen Container brauchen. Kompositionen sind sowohl mit Kardinalitäten von 1 und 0..1 möglich. In unserem Beispiel könnte das etwa bedeuten, dass es gemeindefreie Bezirke geben könnte, so wie wir es von kreisfreien Städten kennen

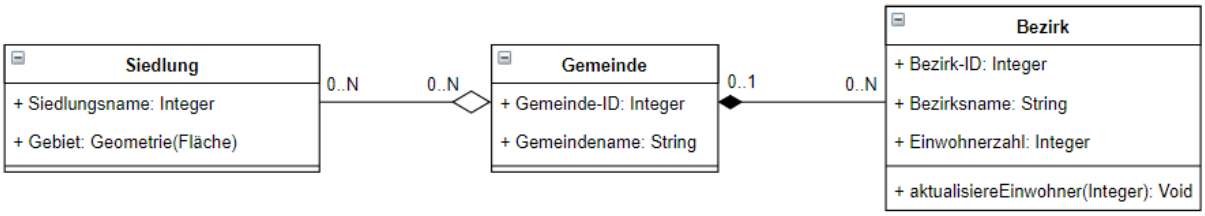

Abbildung 8: Aggregation und Komposition am Beispiel

Damit haben wir alle wichtigen Aspekte des *Klassendiagramms* abgedeckt. Diese Aspekte liegen auch vielen weiteren Diagrammen in der *UML* zugrunde, da hier das objektorientierte Paradigma in seiner Grundform modelliert wird.

Anwendungsfalldiagramm

Zum Abschluss dieser Einführung soll nun auch noch ein *Verhaltensdiagramm* erklärt werden, namentlich das *Anwendungsfalldiagramm*.

In *Verhaltensdiagrammen* werden sowohl das dynamische Verhalten eines *Systems*, als auch die Interaktionen der Nutzer mit diesem modelliert. Bei einem *Anwendungsfalldiagramm* liegt der Fokus auf diesen Interaktionen. Genauer gesagt darauf, welche Nutzer es gibt, und für was sie das *System* nutzen können.

Die Nutzer werden hier als *Akteure* bezeichnet. Im Diagramm sind diese als Strichmännchen gekennzeichnet. *Akteure* werden über *Assoziationen* (ähnlich wie in *Klassendiagrammen*) mit *Anwendungsfällen* verbunden, welche als Ellipsen dargestellt werden. *Akteure* können untereinander über *Generalisierungen* verbunden sein, genau wie *Anwendungsfälle*. Das Konzept ist dabei dasselbe wie bei *Klassendiagrammen. Akteure,* die mit dem generellen *Anwendungsfall* verbunden sind, haben die gleiche *Assoziation* auch zu den spezialisierten *Anwendungsfällen*, und spezialisierte *Akteure* übernehmen die *Anwendungsfälle,* mit denen ihr generalisierter *Akteur* verbunden ist.

Es gibt außerdem zwei neue Beziehungen: Die „include“- und „exclude“-Beziehung. Diese bestehen zwischen *Anwendungsfällen*, und beschreiben wie diese voneinander abhängen. Wird die „include“-Beziehung genutzt, dann ist der inkludierte *Anwendungsfall* (auf den der Pfeil zeigt) notwendige Bedingung für den aufrufenden *Anwendungsfall*. Er muss also jedes Mal durchgeführt werden, wenn ein Nutzer den aufrufenden *Anwendungsfall* aktiviert. Bei der „extend“-Beziehung ist dies kein Muss mehr, sondern optional.

Weiterhin können die *Anwendungsfälle* oft reale oder über das *System* hinausgehende Anforderungen haben. Daher ist es möglich alle systembezogenen *Anwendungsfälle* in einen *Systemkontext* einzuordnen. Dieser wird als Rechteck um die betreffenden *Anwendungsfälle* gezeichnet.

Nachfolgend ein Beispiel, welches sich an unserem „Baumkataster“ orientiert:

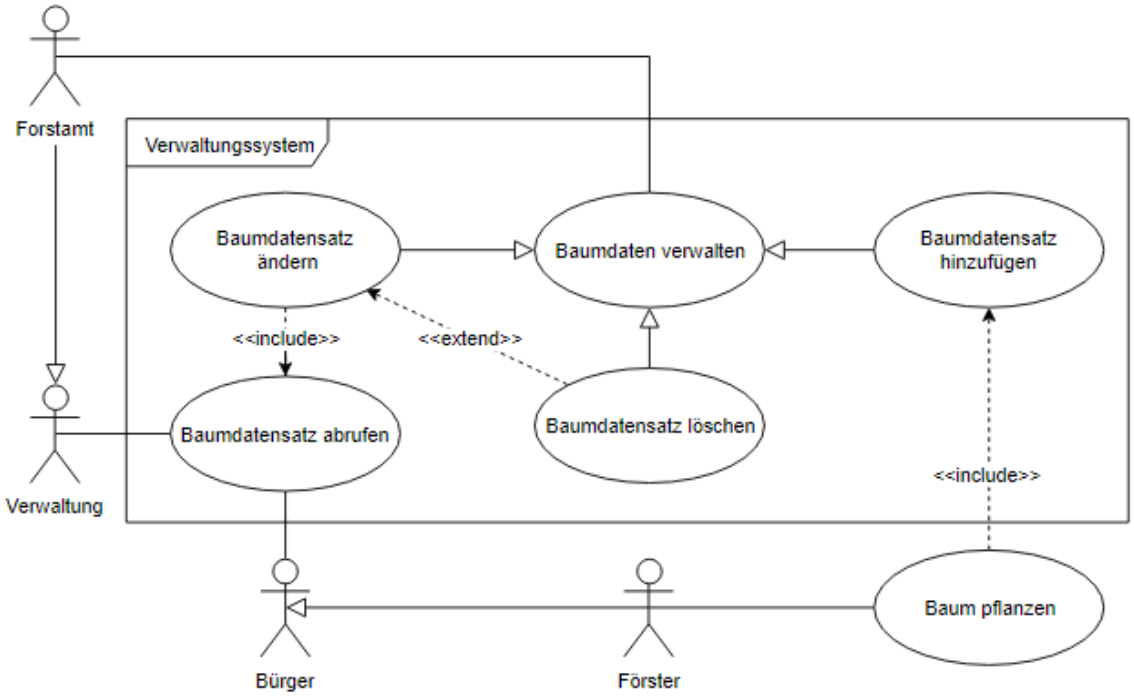

Abbildung 9: Beispiel für ein Anwendungsfalldiagramm

Gehen wir die Elemente Schritt für Schritt durch: Eine Art von *Akteur* ist die Verwaltung, also die Menge an Verwaltungsmitarbeitern. Arbeiten diese nicht im Forstamt, können sie die Daten im Baumkataster nur abrufen. Arbeiten sie im Forstamt, haben sie besseren Zutritt zum System, und können Datensätze ändern, löschen und hinzufügen. Sie können aber trotzdem noch als Verwaltungsmitarbeiter auf die Daten zugreifen, daher sind sie eine *Spezialisierung* des Verwaltungs-Akteurs. Ändern, Löschen und Hinzufügen werden alle als Verwaltungsaufgaben angesehen, und fallen daher unter den generellen *Anwendungsfall* namens „Baumdaten verwalten“. Um Änderungen vorzunehmen, muss erst auf den Datenbestand zugegriffen werden, daher gibt es hier eine „include“-Beziehung. Beim Hinzufügen ist es optional, ob vorher auf den Datensatz zugegriffen wird, etwa um zu prüfen ob der Baum schon vorhanden ist. Daher wurde hier eine „extend“-Beziehung verwendet.

Eine weitere Art von *Akteur* ist dann der Bürger. Dieser kann laut unserem Modell die Daten nur abrufen. Der Förster wird abstrahiert als spezieller Bürger, welcher Bäume pflanzen darf. Wird ein Baum gepflanzt, muss dieser auch in die Daten aufgenommen werden, was durch eine „include“-Beziehung verdeutlicht wird.

Offensichtlich sind hier nicht alle möglichen *Anwendungsfälle* modelliert. Genau das liegt aber im Zentrum der *Modellierung*: Wir abstrahieren auf das Level was benötigt wird. Wollten wir weiter ins Detail gehen, könnte man mehrere Forstamt-*Akteure* spezialisieren, die verschiedene Aufgaben übernehmen. Man könnte auch das Fällen von Bäumen modellieren, wo sich dann die Frage stellt, ob der Datensatz sofort gelöscht werden oder nur ein Status auf „gefällt“ geändert werden sollte.

Ausblick

UML bietet noch viele weitere Diagramme, welche an dieser Stelle nicht mehr erklärt werden sollen. Interessierte Leser können tiefergehende Links im Quellenverzeichnis finden. Abseits von Diagrammen bietet UML noch weitaus mehr, teils hochkomplexe Strukturen. So können für stark formalisierte Systeme die Diagramme untereinander kombiniert werden, wobei alle Diagramme auf den gleichen

Pool an Elementen (*Klassen*, *Akteure*, *Anwendungsfälle*, etc.) zugreifen und diesen aus unterschiedlichen Winkeln beleuchten. Will man auf diesem Level modellieren, dann nutzt man für gewöhnlich spezialisierte Tools, welche erlauben die Modelle und Diagramme direkt in austauschbare Formate zu exportieren.

Arbeit man nah genug an Standards und Formalismen, gibt es sogar Ansätze, welche versuchen Software direkt aus UML-Spezifikationen herzuleiten. Wer die Macht, aber auch die Komplexität von *UML* im Geodaten-Bereich in Aktion erleben möchte, der schaue sich die **Infrastructure for Spatial Information in the European Community** (*INSPIRE*) oder das ***AAA-Modell*** an, welche durchgehend mit *UML* modelliert wurde.

Literatur

Bill, R. (2016): Grundlagen der Geo-Informationssysteme. 6. Auflage. Wichmann Verlag. Offenbach-Berlin. 867 Seiten. Kapitel 4.2.

Booch, G., Rumbaugh, J., Jacobson, I. (1999): Das UML-Benutzerhandbuch. Addison-Wesley. 592 Seiten.

Internet

Ein Hinweis: Es gibt zahllose Quellen zur UML-Modellierung, aus verschiedenen fachlichen Gesichtspunkten. Viele dieser Quellen (selbst Wikipedia!) enthalten jedoch formale Fehler in ihren Diagrammen. Außerdem gibt es historisch gewachsen viele verschiedene Notationsstandards. Für die meisten Anwendungen ist es nicht wichtig ob Formalismen genau beachtet werden, solange die Diagramme ihren Zweck erfüllen. Ist das Ziel jedoch ein formal perfektes UML-Diagramm, dann sollte stets die offizielle Dokumentation zu Rate gezogen werden, so undurchsichtig sie auch sein mag.

https://www.omg.org/spec/UML/ - Aktuelle Spezifikation

https://de.wikipedia.org/wiki/Unified_Modeling_Language - Übersichtsartikel

https://de.wikipedia.org/wiki/Entity-Relationship-Modell - Siehe vor allem verschiedene Notationsformen

http://inspire.ec.europa.eu/ - INSPIRE