# 5 Simple Feature Access Part 1: Common Architecture

## 5.1 Überblick über die Inhalte der Norm

Wie wir gesehen haben, setzt Teil 1 der *Simple-Feature*-Spezifikation, die *Common Architecture* – kurz „SFA-CA" –, die Konzepte des *Spatial Schema* für die Untermenge einfacher geometrischer Klassen wie Points, LineStrings und Polygone sowie ihrer Aggregate um. Ab der Version 1.2.0 schließt das „erweiterte Geometriemodell" der *Simple Features* auch Polyeder, triangulierte Netzwerke und *MultiSurfaces* ein [SFA-CA].

Die *Common Architecture* bewegt sich noch auf der konzeptionellen Ebene. Sie beschreibt in UML das gemeinsame Geometriemodell der *Simple Features*, das in den Implementierungsvarianten in konkrete Klassen und Signaturen umgesetzt wird. Daneben definiert sie Codierungen, die alle Varianten verwenden. Sie besteht aus drei Teilen:

- Das *Geometry Model* definiert die geometrischen Grundformen der *Simple Features* mit ihren Attributen und Methoden. Dazu gehört auch das gemeinsame Verständnis für relationale Operatoren, mit denen die räumliche Beziehung zwischen Geometrien geprüft und eine Geometriemenge gefiltert werden kann (hier Kapitel 5.3 und 5.10).
- Neben dem Geometriemodell definiert die *Common Architecture* ab Version 1.2.0 auch eine Struktur für Beschriftungen (*Annotations*). Mehr dazu im Kapitel 5.12.
- Mit den ***Well-known Structures*** definiert die SFA-CA allgemeine Übergabeformate für Geometrien und Koordinatensystem-Information in Form von ***Well-known-Text-*** und ***Well-known-Binary***-Repräsentationen (Kap. 5.7).

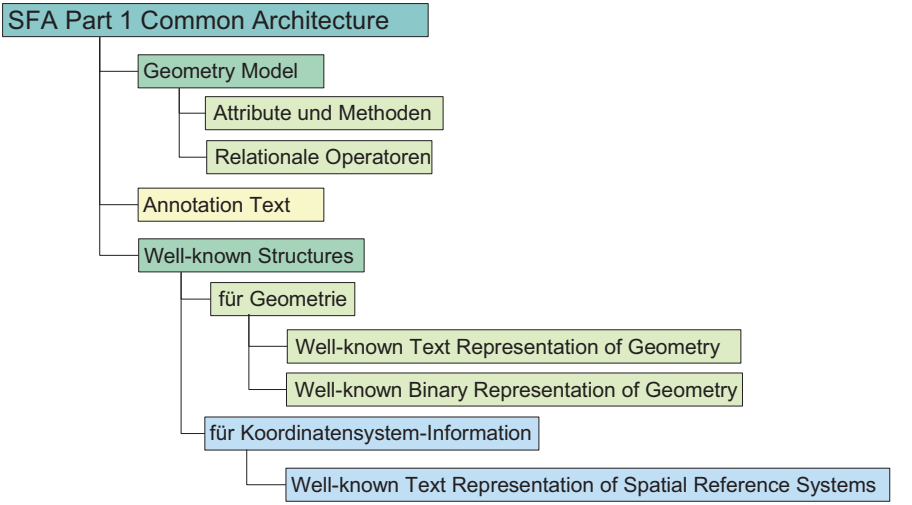


**Abbildung 23: Gliederung der *Common-Architecture*-Norm (SFA-CA) in Version 1.2.0. Gelbe Teile gibt es erst ab Version 1.2.0; grüne Teile wurden gegenüber der Version 1.1.0 verändert.**

## 5.2 Grundlagen: Objektorientierte Modellierung und UML

Die grundlegenden Modelle der OpenGIS-Welt ebenso wie die konzeptionellen ISO-Normen für den Bereich der Geoinformatik folgen dem Denkansatz der objektorientierten Modellierung. Diese Art des Softwareentwurfs wurde in den 80er-Jahren erfunden und setzte sich in den 90er-Jahren zunehmend in der Softwareentwicklung durch. Alle OpenGIS-Standards und die ISO-Standards der 19000er-Reihe verwenden objektorientierte Sichtweisen für die Beschreibung der zugrunde liegenden Konzepte.

Während wir in dem Vorläufer der objektorientierten Programmierung, der „strukturierten Programmierung" von Routinen, Prozeduren und Funktionen sprechen, geht es in der objektorientierten Programmierung um Objekte, (Objekt-)Klassen, Methoden und Attribute und ihre Bündelung in Komponenten oder Module ([LAHRES]). Der Vorteil der objektorientierten Programmierung liegt in der flexiblen Wiederverwendbarkeit einmal geschriebenen Codes und dem Aufbau modularer Anwendungen daraus.

Ein **Objekt** ist ein Programm- oder Datenelement und die Grundeinheit der objektorientierten Programmierung. Ein Objekt kann sowohl Daten enthalten als auch Funktionalität zur Verfügung stellen. Es gehört immer einer (Objekt-)**Klasse** an. Die Klassendefinition legt fest, welche Eigenschaften (***Properties***) die Objekte der Klasse haben können. Zu den Eigenschaften gehören Attribute, Operationen bzw. Methoden und Beziehungen zu anderen Klassen.

Daten sind in den **Attributen** eines Objekts enthalten. Dies können Werte sein, wie z. B. die Koordinaten einer Geometrie, oder Informationen über den aktuellen Zustand des Objekts. Es können auch Werte sein, die das Objekt beim Abruf erst neu berechnet. Die Namen und Datentypen der Attribute werden mit der Klasse definiert, der ein Objekt angehört. Die Daten eines Objekts können in der Regel nicht direkt, sondern nur über seine Methoden geändert werden (**Prinzip der Kapselung**).

Über seine **Operationen** stellt ein Objekt Funktionalität – ggf. zur Änderung seiner Dateninhalte – für die Nutzung zur Verfügung. Die „Signatur" einer Operation ist deren Beschreibung; sie wird mit der Klasse festgelegt und besteht aus dem Namen der Operation, ggf. notwendigen Aufrufparametern und der Art der zurückgegebenen Werte oder Objekte.

Beispielsweise dient die Operation *srid* eines Geometrieobjekts dazu, die Koordinatenwerte der Geometrie einem anderen Bezugsystem zuzuordnen; sie benötigt als Eingabeparameter einen Identifikator für das neue Bezugssystem und gibt ein Geometrieobjekt mit geändertem Bezugssystem zurück. Die Operation *area* flächenhafter Geometrieobjekte soll den Flächeninhalt ermitteln und ausgeben.

**Methoden** sind konkrete Implementierungen dieser Operationen. Wie der Algorithmus hinter einer Methode genau aussieht, ist der Implementierung hinter den Kulissen vorbehalten. Wie beispielsweise eine *area*-Methode in der konkreten Realisierung den Flächeninhalt eines Polygonobjekts ermittelt, ist ihr selbst überlassen. Das hat den großen Vorteil,

dass hinter der „Maske" des Methodenaufrufs der eigentliche Programmcode angepasst und verändert werden kann. Gleichnamige Methoden können auch für verschiedene Klassen unterschiedliche Algorithmen haben; so wird der Code für die *area*-Methode eines Multipolygons etwas anders ausfallen als der Code für die *area*-Methode eines Polygonobjekts (**Prinzip der Polymorphie**).

Ein Objekt kann **Beziehungen** (Assoziationen) zu anderen Objekten haben und nimmt dann in diesen Beziehungen eine bestimmte **Rolle** ein. Eine **Aggregation** ist eine Zusammensetzung eines Objekts aus Objekten einer anderen Klasse. Aggregationen können noch weiter differenziert werden: Eine Aggregation im engeren Sinne setzt sich aus Objekten zusammen, die unabhängig von der Aggregation existieren können. Dagegen besteht eine **Komposition** aus Teilobjekten, die ohne die Komposition keine eigene Existenz haben. Auch die möglichen Beziehungen zu Objekten anderer Klassen und deren Rollen sind Bestandteil der Klassendefinition.

Wie viele Beziehungspartner die Objekte einer Klasse in einer Rolle oder wie viele Ausprägungen ein Attribut maximal haben kann, das wird durch die **Kardinalität** der Eigenschaft beschrieben. In der Regel sind das Werte wie 1 oder n. Die **Multiplizität** einer Rolle oder eines Attributs gibt die Spannbreite von der minimalen bis zur maximalen Anzahl der Beziehungspartner oder Werte an. Beziehungen können ungerichtet oder gerichtet sein. Gerichtete Assoziationen können nur in einer Richtung „navigiert", das heißt ausgewertet werden: Objekte der einen Klasse „kennen" dann ihre Beziehungspartner, aber nicht umgekehrt.

Sind Objekteigenschaften „*public*", dann können andere Objekte sie abfragen oder aufrufen. Dagegen sind „*private*" Eigenschaften von außen nicht sichtbar. OpenGIS-Modelle definieren in der Regel nur Eigenschaften, die „*public*" sind.

Ein einzelnes Objekt einer Klasse ist eine „**Instanz**" seiner Klasse; im Deutschen wird auch der Ausdruck „Exemplar" dafür verwendet ([LAHRES]). Eine Instanz ist immer ein Individuum, eine konkrete, in irgendeinem Speicher verwaltete Menge an Bits und Bytes. Ihre Attribute haben konkrete Werte und ihre Beziehungen verbinden es mit konkreten anderen Instanzen. Eine einzelne Instanz hat eine begrenzte Lebensdauer und häufig eine eindeutige Identität. Sie wird mithilfe eines sog. **Konstruktors** erzeugt. Eine Instanz kann – je nach Implementierung – evtl. auch mehreren Klassen angehören.

Klassen können Ober- und Unterklassen haben. Unterklassen übernehmen alle Eigenschaften (Attribute, Methoden, Rollen) ihrer Oberklassen (**Prinzip der Vererbung**). Sie sind in der Regel Spezialisierungen ihrer gemeinsamen Oberklassen, d. h., sie unterscheiden sich von ihren Geschwisterklassen durch differenzierende Merkmale. Das können besondere Attribute, Methoden oder Beziehungen sein. Manchmal müssen die Instanzen einer Unterklasse aber auch nur bestimmte Bedingungen erfüllen; beispielsweise müssen bei der

Geometrieklasse *LinearRing* Anfangs- und Endpunkt zusammenfallen; die Klasse hat aber sonst die gleichen Eigenschaften wie ihre Oberklasse *LineString*. Durch das Vererbungsprinzip bilden Klassen in einem Objektmodell meist einen hierarchischen Baum. Die Beziehung zwischen Unterklasse und Oberklasse nennt man auch **Generalisierung**.

Objektklassen können „abstrakt" oder „instanzierbar" sein. **Abstrakte Klassen** sind lediglich Zwischenebenen in der Klassenhierarchie, in der gemeinsame Eigenschaften ihrer Unterklassen gebündelt sind. Es dürfen jedoch keine Objektindividuen einer abstrakten Klasse gebildet werden, sondern Objekte müssen immer einer der Unterklassen angehören. Klassen, zu denen konkrete Objekte erzeugt werden können, heißen „instanzierbar".

Die objektorientierte Programmierung ermöglicht die flexible Nutzung von Codebausteinen. Als Grundbausteine verwendet sie Objekte, die immer einer definierten Klasse angehören. Zu einer Klasse werden Attribute, Methoden und Rollen definiert, die dann in Instanzen der Klasse realisiert werden.

Für diese Grundbausteien gelten drei wesentliche Prinzipien: In Objekten verwaltete Dateninhalte sind grundsätzlich nur über die Methoden der Objekte zugänglich (Kapselung). Spezialisierungen haben alle Eigenschaften ihrer Oberklasse (Vererbung). Und eine Methode kann für Objekte verschiedener Klassen unterschiedlich realisiert sein (Polymorphie).

Eng mit der objektorientierten Sichtweise verbunden ist die Dokumentation des konzeptionellen Modells in der ***Unified Modeling Language*** *(*UML). Auch die neueren *Simple-Feature*-Spezifikationen verwenden diese Notationsmethode für die Beschreibung des Geometriemodells.

In UML gibt es verschiedene Arten grafischer Diagramme für das Entwerfen von Softwarekomponenten und -architekturen. Die am häufigsten verwendete Diagrammart ist das **statische Klassendiagramm**. Es beschreibt (Objekt-)klassen und deren Beziehungen sowie ihre Attribute, Methoden und Beziehungsrollen. Die wichtigsten hier vorkommenden Notationsregeln sind im darauf folgenden Textkasten zusammengestellt (Abb. 24).

Sämtliche Klassen der *Simple Features* sind in diesem Buch auch in Klassendiagrammen vertreten. **Farben** in den Klassendiagrammen in diesem Buch stammen nicht aus den Originaldokumenten und haben keine inhaltliche Bedeutung, werden aber hier zur Strukturierung zugunsten von mehr Übersichtlichkeit eingesetzt.

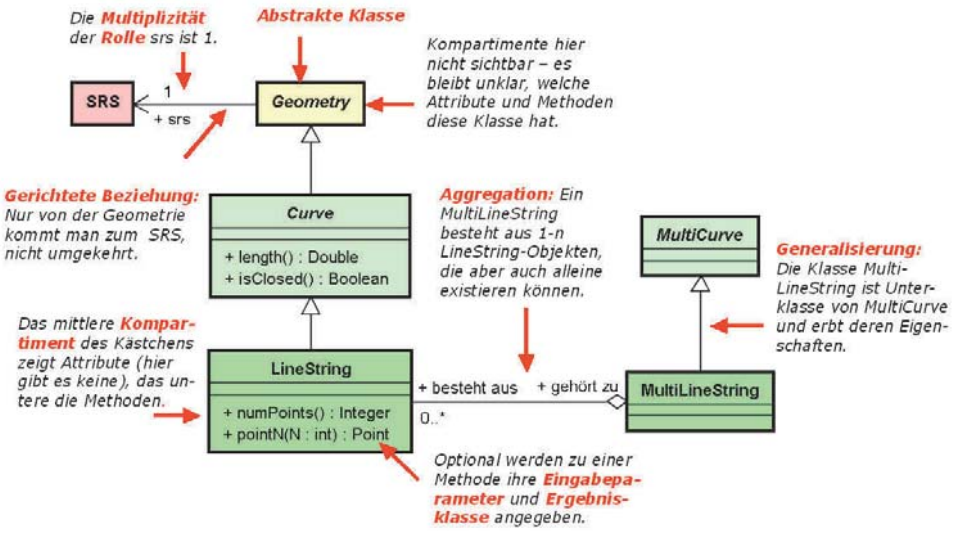


**Abbildung 24: Beispiel für ein UML-Klassendiagramm mit den wichtigsten Elementen**

**Notationsregeln für Klassendiagramme in UML**

**Klassen** werden als Kästchen dargestellt, die den Klassennamen tragen. Bei abstrakten Klassen wird der Klassenname kursiv gesetzt. Klassennamen beginnen immer mit einem Großbuchstaben.

**Attribute** und **Methoden** können bei Bedarf in eigenen Kompartimenten des Kästchens aufgelistet werden. Optional können bei Attributen der Datentyp und bei Methoden eventuelle Eingabeparameter und der Datentyp bzw. die Objektklasse des Rückgabewertes angegeben werden.

Die Namen von Attributen, Methoden oder Rollen (i. e. *Property*-Namen) beginnen grundsätzlich mit einem kleinen Buchstaben. Eine Ausnahme bilden die **Konstruktormethoden** für die Erzeugung von Objekt-Instanzen. Sie tragen meist den Namen der Klasse. In der Regel ist die Konstruktormethode die einzige Methode, deren Namen großgeschrieben wird.

Methodennamen werden häufig durch ein nachgestelltes Klammerpaar von Attributen unterschieden.

Ein „+" vor Attribut-, Methoden- oder Rollennamen bedeutet, dass diese Eigenschaft nach außen hin für andere Objekte sichtbar („*public*") und verwendbar ist.

**Beziehungen** werden im Klassendiagramm mit einer durchgezogenen Linie dargestellt; bei gerichteten Assoziationen zeigt ein Pfeil die mögliche Navigationsrichtung an. In den Klassendiagrammen wird die Multiplizität beispielsweise in der Form 0, 0..1 oder 1..* angegeben, wobei * für eine unbegrenzte Anzahl steht.

Beziehungsnamen beginnen mit Großbuchstaben und werden im Klassendiagramm etwa mittig über oder unter die Beziehungslinie gesetzt.

Die Rollen einer Beziehung werden in Kleinbuchstaben an die Enden der Beziehungslinie gesetzt.

**Aggregationsbeziehungen** werden im Klassendiagramm durch ungefüllte Rauten am Beziehungsende, **Kompositionen** durch schwarz ausgefüllte Rauten dargestellt.