Das Objektorientierte Paradigma

Embedded Software Engineering Report Nr. 30

WILLERT.

Inhalt:

Das Objektorientierte Paradigma, eine Einlassung aus persönlicher Perspektive

„Nichts ist schwieriger als das Vereinfachen.
Nichts ist einfacher als das Komplizieren."
Georges Elgozy (Politiker und Autor, 1909-1989)

Ich habe für meine Studenten lange nach einem einfachen Erklärungsmodell der Objektorientierung gesucht. Mir schwebte etwas vor wie das EVA-Leitbild für die Modulidee. Einprägsam und doch so mächtig, dass sich daraus alles Wesentliche herleiten ließe. Es hat fünfzehn Jahre gedauert bis ich das objektorientierte Paradigma in einen Satz packen konnte.

„Wir geben den Dingen Namen!"
Dipl. Ing. Päd. Alexander Huwaldt

Es ist tatsächlich so erschreckend einfach und ich hoffe Ihnen mit meiner Einlassung zu diesem Thema einige Anregungen geben zu können. Die Auseinandersetzung mit der Objektorientierung stellt für Einsteiger und für Umsteiger oft eine gewisse mentale Herausforderung dar. Auch die Strukturierung wurde am Anfang von den Entwicklern abgelehnt, weil die Freiheitsgrade der Gestaltung von Algorithmen stark eingeschränkt wurden. Manchmal fußte die Ablehnung auch einfach darauf, dass das Basiskonzept der Block- / Modulbildung nicht verinnerlicht wurde und man sich in Schimpftiraden über die verdammten Klammern erging. Die Strukturierung baute jedoch Komplexität ab. Das Paradigma der Strukturierung war sinngemäß der Schritt vom Metallbaukasten zu den Legobausteinen, von der Freiverdrahtung zum Hutschienenmodul. Dies förderte eine schnelle und dauerhafte Akzeptanz der Strukturierung.

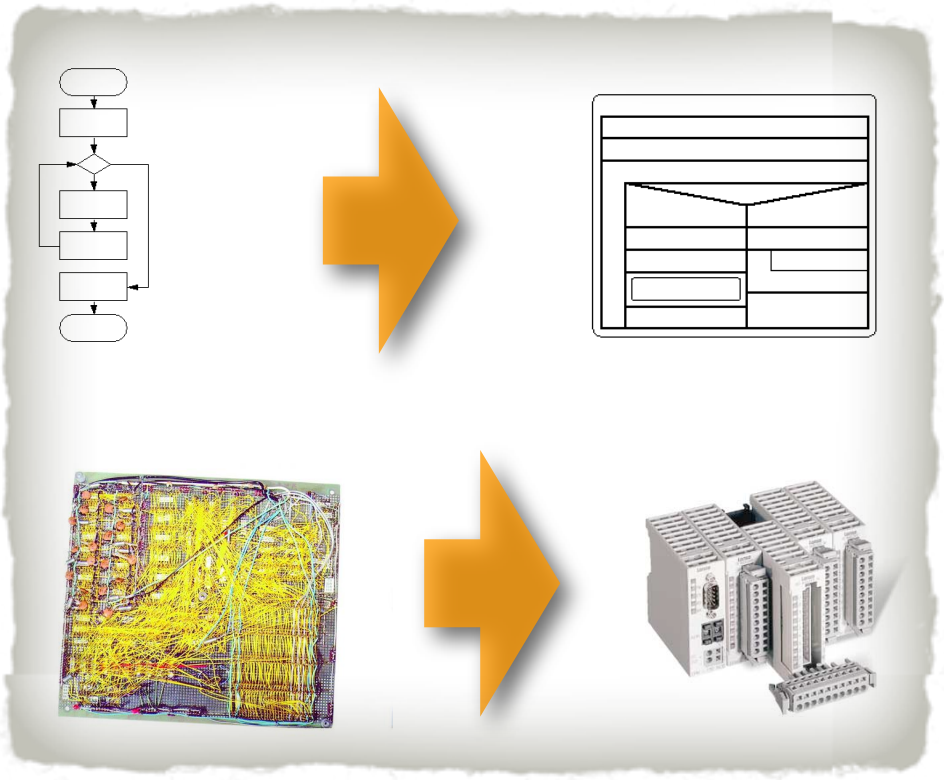


Abbildung 1: *Von der Freiverdrahtung zum Hutschienenmodul*

Im Gegensatz zur Strukturierung stellt die Objektorientierung kein vereinfachendes, einschränkendes Paradigma, sondern ein komplexes, fast schon philosophisches Theoriegerüst dar. Das macht den schnellen Zugang schwer. Oft wird in die Hülle einer objektorientierten Sprache nur mehr oder weniger strukturiert hinein programmiert. Man erkennt dies zum Beispiel an Methoden mit hundert oder mehr Zeilen, vielen Entscheidungen, keine oder nur sehr spärlich genutzte Vererbung und Polymorphie usw. Der schwere Zugang zur Objektorientierung erklärt auch, warum diese erst so spät Akzeptanz gefunden hat. Sie ist eigentlich genauso alt, wie die Strukturierung und es gab sogar zuerst eine objektorientierte Sprache, bevor es eine echte strukturierte Programmiersprache gab.

Entstehung ausgewählter Programmiersprachen

1940er- 1970er	Maschinensprache, Assembler, Sprungorientierung
1944	Plankalkül (Zuse)
1950er	Fortran, …
1960er	Cobol, Algol, BASIC, …
1967	Simula 67: Objektorientierung
1971	Pascal: Strukturierung
70er,80er,90er:	C, C++, Java, C#, …

Wenn man sich das Wesen der Entwicklung vergegenwärtigt, erkennt man, dass uns die Suche nach dem Ausgangspunkt der Programmiersprachen zum sequenziellen Automat und zur Turingmaschine führt. Diese Ursprünge bilden sich in der tatsächlichen Funktionsweise, dem anfänglichen Programmierkonzept und der Maschinensprache ab. Das herausragende Konzept der Turingmaschine ist, dass diese spezielle Anweisungen besitzt, den programmgesteuerten Lesekopf auf einem Befehlsband aus dem Programm heraus zu positionieren. Diese Sprunganweisung wurde das Basiskonzept für die Funktionsweise von Digitalrechnern und

damit auch der anfänglichen Programmiersprachen. Ab dem nächsten Entwicklungsschritt ist eine zunehmende Abstraktion vom wirklichen Funktionsprinzip der Maschine auf immer höherem Niveau zu beobachten. Von der Maschinensprache im Binärcode über die Assemblersprache wurde zunehmend abstrahiert zu Hochsprachen, mit leistungsfähigen Befehlen und komplexer Grammatik. Diese können unter Umständen hunderte Maschinenbefehle in einer Zeile zusammen fassen. Der nächste Schritt war, vom Funktionsprinzip des Sprungs zu abstrahieren und jegliche Programmierung auf die Verwendung von drei algorithmischen Bausteinen (Sequenz, Iteration, Alternative) zu reduzieren. Der nächste Abstraktionsschritt entfernt sich noch weiter vom Konzept der programmierbaren Maschine, die Eingaben verarbeitet und daraufhin Ausgaben ausführt. Die im Rechner abgebildeten Programme sollen nicht mehr den tatsächlichen Funktionsprinzipien eines Digitalrechners folgen, sondern in Struktur und Verhalten einem abgebildeten Realitätsausschnitt entsprechen. Das ist die Objektorientierung.

„Jede neue Sprache ist wie ein offenes Fenster, das einen neuen Ausblick auf die Welt eröffnet und die Lebensauffassung weitet."
Frank Harris (Schriftsteller 1856-1931)

Daniel Ingalls, einer der Urväter der objektorientierten Programmierung und Entwickler der Sprache SMALLTALK formulierte 1981 in seinem Zeitungsartikel „Design Principles Behind Smalltalk" die Grundidee der Objektorientierung sinngemäß wie folgt:

„Bei der Gestaltung einer Sprache für die Verwendung mit Computern haben wir nicht lange nach hilfreichen Tipps suchen müssen. Alles, was wir darüber wissen, wie Menschen denken und kommunizieren, ist dafür anwendbar. Die Mechanismen menschlichen Denkens und menschlicher Kommunikation haben sich über Millionen von Jahren entwickelt und wir sollten dies als gegebenen Sprachentwurf respektieren. Darüber hinaus müssen wir übrigens mit diesem Entwurf der Natur die nächsten Millionen Jahre auskommen. Es spart einfach Zeit, wenn wir unsere Computermodelle kompatibel mit unseren Gedanken machen, statt umgekehrt."

Als Basiskonzepte objektorientierter Computermodelle gelten im Wesentlichen:
Abstraktion, Objekt, Klasse, Generalisierung, Kapselung, Nachricht, Polymorphie, Assoziation, Aggregation und Komposition. Ausgehend von der Idee, dass die Gestaltung objektorientierter Computermodelle ihren Ausgangspunkt in der Betrachtung der Realität sowie der menschlichen Art und Weise zu denken und zu sprechen hat, sollen diese Basiskonzepte kurz umrissen werden. Objektorientierte Programmiersprachen zeichnen sich letztlich dadurch aus, für diese zunächst abstrakten Konzepte konkrete Ausdrucksmittel anzubieten. Wir müssen uns nur darüber bewusst werden wie wir denken und sprechen.

Abstraktion

lat. abstractus „abgezogen", von *abs-trahere* „abziehen, entfernen, trennen"
Bedeutung: von der Gegenständlichkeit losgelöst, Verallgemeinerung, vgl. Generalisierung, Vereinfachung, engl. abstraction

Den Ausgangspunkt der Objektorientierung bildet die Annahme, dass Computerprogramme Sachverhalte der realen Welt abbilden sollen, um dem Menschen die Arbeit zu erleichtern oder auf irgendeine Art dienstbar zu sein. Das Verstehen und die Beschreibung der realen Welt reflektieren sich über die menschliche Sprache. Um einem Computer einen bestimmten Sachverhalt als Programmlogik zu vermitteln, soll sich dieser Prinzipien bedient werden. Der Weg, welcher nachzuzeichnen ist, führt vom betrachteten Realitätsausschnitt (Diskurs) über die Wahrnehmung (sehen, hören, fühlen, schmecken, riechen) zum Verstehen und zur Sprache. Eine Schlüsselfunktion kommt hierbei der Bildung von Worten bzw. Begriffen zu. Ein wahrgenommener Sachverhalt muss in Worte gefasst werden, um diesen zu verstehen und zu beschreiben. Die Begriffsbildung ist der Schlüssel zum Verständnis der Abstraktion.
Vereinfacht ausgedrückt passiert folgendes: Wir sehen in einem Raum, unserem Diskurs, Dinge auf denen man sitzen kann. Diese ähneln sich sehr und stehen mit anderen Dingen und uns im Zusammenhang. In unserem Denken verknüpfen wir die wahrgenommenen Dinge mit dem erlernten Begriff „Stuhl". Der Begriff „Stuhl" aber ist abstrakt. Er existiert losgelöst vom konkreten Gegenstand. Wir können den Begriff nicht nur auf einen, sondern auf viele gleichartige Dinge anwenden und wir können über Stühle nachdenken und kommunizieren, ohne dass ein Stuhl da sein muss. Voraussetzung ist, dass wir den Begriff erlernt und verstanden haben. Das Erlernen und Verstehen des Begriffs „Stuhl" erfolgte wiederum über die

Wahrnehmung von Stühlen und der Verknüpfung der wahrgenommenen gemeinsamen und allgemeinen Merkmale (Beine, Sitzfläche, Lehne, etc.) mit dem künstlich geschaffenen Begriff. Dabei werden unwesentliche oder individuelle Merkmale einzelner Stühle von der Begriffszuordnung ausgeschlossen und der Begriff wiederum aus Begriffen zusammengesetzt. In unserem Denken liegt eine Verknüpfung von allgemeingültigen Merkmalen, man könnte fast sagen Bauplänen und Begriffen, vor. In der Sprache und Begriffsbildung manifestiert sich die menschliche Fähigkeit zur Abstraktion.

Objekt

lat. obiectum „das Entgegengeworfene“
Bedeutung: Gegenstand, auf den das Denken o. Handeln ausgerichtet ist, Ding, Sache, Ausprägung, vgl. Subjekt, Instanz und Entität
engl. object

Beim Erfassen der Realität treten uns Sachverhalte mit wahrnehmbaren Merkmalen entgegen (Farbe, Größe, Gewicht, Temperatur, usw.). Sind in einem Diskurs klare Grenzen zu erkennen und bilden diese abgeschlossene Einheiten (Instanzen), so können wir diese als Gegenstände (Dinge) wahrnehmen. Gegenstände besitzen also klare Grenzen und verfügen über Merkmale. Die wahrgenommenen Merkmale betreffen die Struktur der Gegenstände (Eigenschaften), aber auch die Veränderung der Merkmale (Verhalten). Da sich unser Denken und Tun beim Wahrnehmen eines Gegenstandes auf diesen bezieht, nennt der Erkenntnistheoretiker diesen „Objekt der Erkenntnis“. Wir selbst sind in diesem Kontext das „Subjekt der Erkenntnis“. Die gewonnene Erkenntnis versetzt uns in die Lage, ein Urteil, eine Aussage (lat. prädicatum) über das Objekt zu tätigen. Um mit dem Gegenstand im Denken umzugehen, geben wir diesem einen Namen. Hier schließt sich der Kreis zur Abstraktion, Begriffsbildung und Sprache. „Begriff“ leitet sich übrigens davon ab, etwas zu berühren, zu begreifen. Für die wahrgenommenen Objekte finden wir Begriffe und verwenden diese in der Sprache als Namen der Objekte. Der Sprachwissenschaftler bezeichnet Begriffe als „Substantive“ und deren Verwendung im Satzbau als „Subjekt“ und „Objekt“. Die getroffene Aussage, die sich auf das Subjekt bezieht oder dieses mit anderen Objekten in Beziehung setzt, wird als „Prädikat“ bezeichnet.

Es lassen sich unter anderem folgende Thesen zum Objekt aufstellen:

- Systeme bestehen aus abgrenzbaren Objekten
- Objekte spielen eine bestimmte Rolle im System, sie haben eine Bedeutung
- Objekte besitzen Merkmale, diese können unterschieden werden in:
  - Strukturmerkmale (Eigenschaften, Attribute, Aufbau)
  - Verhaltensmerkmale (Verhalten, Funktion, Operationen, Methoden)
- Objekte sind Träger von Informationen und sind autonom (haben alles an Merkmalen bei sich, um ihre Rolle zu erfüllen)
- Objekte haben einen Lebenszyklus: entstehen/erzeugen, existieren, vergehen/zerstören
- Objekte besitzen eine Identität, unterscheiden sich in der Summe ihrer Merkmale von jedem anderen Objekt, sind einzigartig
- Objekte sind vollständig und autonom, sie sind selbst Träger aller Merkmale (Eigenschaften und Verhalten), die ihre Bedeutung ausmachen
- Objekte besitzen Zustände, ein Zustand bezeichnet die Art und Weise, wie etwas zu einem bestimmten Zeitpunkt ist, Zustände werden durch eine endliche Menge konkreter Merkmale (Eigenschaften und Verhalten) charakterisiert
- Objekte stehen zu anderen Objekten im System in Beziehung

Betrachtet man die Begriffsbildung näher, so lässt sich jedoch feststellen, dass ein Begriff in der Regel nicht eine einzelne Ausprägung bezeichnet, sondern wie oben beschrieben eine Menge von gleichartigen Ausprägungen abstrahiert.

Klasse

lat. classis „Herbeirufung", „mil. Abteilung" von calare = einberufen, herbeirufen

Bedeutung: Gruppe mit gemeinsamen Merkmalen, vgl. Kategorie, Menge, Typ

engl. class

Was passiert, wenn wir diese soeben erkannten Objekte abstrahieren? Wir lösen die erkannten Merkmale von ihrem „Untergrund" und bilden diese in unseren Gedanken ab. Das machen wir ununterbrochen. Wir sehen, hören, riechen etwas und haben es im Kopf und können uns daran erinnern. Das versetzt uns in die Lage, auch in Abwesenheit der realen Dinge, diese in unserem Denken zu benutzen. Eine besondere Gabe des Menschen ist es, dass wir neben der gefühlten Abbildung (wir haben Gedächtnisse für Bilder, für Gerüche, für Töne usw.) zu Wahrgenommenem eine Folge von stark differenzierten Lauten assoziieren können. Wir sind in der Lage, den wahrgenommenen Merkmalen ein faktisch willkürlich erfundenes Wort zuzuordnen und dieses in Form von Sprache weiter zu geben. Das Wahrgenommene sind in der Summe die Struktur- und Verhaltensmerkmale der Dinge, also deren Baupläne. Die mit den Bauplänen assoziierten Worte sind die Namen der Dinge. Übrigens lernen unsere Kinder auch Fremdsprachen auf diesem Weg. Die Englischlehrerin hält eine Karte mit dem Bild eines Hundes hoch. Die Schüler rufen aus ihrem Gedächtnis den deutschen Begriff ab. Dann dreht die Lehrerin die Karte um und der englische Begriff steht auf der Rückseite (Bauplan-Begriff-Sprache). Sie spricht die englische Folge von Lauten aus. Die Kinder sprechen nach. Es ist kein realer Hund im Klassenraum. Es ist sogar noch spannender. Der Hund auf dem Bild ist schwarz und groß. Aber alle Kinder wissen bereits aus Erfahrung, es gibt auch kleine braune Hunde. Hund ist offensichtlich nicht der Name für ein bestimmtes Objekt, sondern ein Begriff für eine Menge von Objekten, die vier Beine und einen Schwanz haben, sowie ab und an bellen. Die vier Beine und der Schwanz sind die gemeinsamen Strukturmerkmale dieser Gruppe von Objekten. Das Bellen ist ein gemeinsames Verhaltensmerkmal. Was hier so unbewusst im Lernprozess angewendet wurde, ist, dass Lehrerin und Schüler sich eigentlich über eine sogenannte Kategorie (griech. **κατηγορία**, das Ergebnis einer Zusammenfassung) verständigt haben. Das Bilden von Kategorien nennt der Fachmann Klassifizierung. Und damit sind wir beim Klassenbegriff. Die Begriffe mit denen wir die Welt beschreiben (Stuhl, Tisch, Schrank, Hund und Katze) repräsentieren in der Regel eine Menge von Objekten mit gemeinsamen Merkmalen. Damit wird das Prinzip der Abstraktion auf das Objekt angewendet und führt uns zum Klassenbegriff.

Für Klassen lassen sich unter anderem folgende Thesen aufstellen:

- Klassen repräsentieren eine Menge von Objekten mit gleichen Merkmalen
- eine Klasse beschreibt die gemeinsamen Merkmale der ihr zugehörigen Objekte (Bauplan mit Eigenschaften und Verhalten)
- Klassen erhalten einen Namen
- Klassen typisieren Ausprägungen, sind also selbst Typen (lateinisch typus von griechisch **τυπος** „vom Schlage", „Gepräge")

Das Hunde ihre Jungen säugen, assoziieren wir nicht sofort mit dem Bild vom Hund. Denn auch Katzen, Mäuse und Menschen säugen ihre Jungen. Es scheint kein dem Hund ausschließlich eigenes Merkmal zu sein. Diese Überlegung führt

uns zu einem weiteren Konzept der Abstraktion in unserem Denken und Sprechen.

Generalisierung

lat. generalis „zum Geschlecht gehörend" von gens „Familie", „Stamm" ***Bedeutung: Verallgemeinerung, zusammenfassen vgl. Vererbung, Spezialisierung, Individualisierung, Abstraktion, Wiederverwendung***
engl. generalization

Führen wir das oben angesprochene Beispiel mit dem Säugen fort, kommen wir zur Überkategorie der Säugetiere. Dabei sind alle den Säugetieren gemeinsamen Merkmale einer übergeordneten Kategorie zugeordnet. Die untergeordneten Klassen (Begriffe) übernehmen die Merkmale der übergeordneten Kategorie (Oberbegriff, Basisklasse) einfach durch ihre Zugehörigkeit. Ein Hund ist ein Säugetier. Dieses Übernehmen der Merkmale nennt man in der Objektorientierung Vererbung. Die dabei entstehende Hierarchie von Klassen (Dogge - Hund – Säugetier – Wirbeltier - … - Lebewesen) kann in zwei Richtungen gelesen werden, vom Allgemeinen zum Speziellen und vom Speziellen zum Allgemeinen. Die eine Lesart nennt man Spezialisierung, die andere Generalisierung. Abstraktion kann somit nicht nur auf Instanzen der Realität angewendet werden, sondern auf die abstrahierten Kategorien selbst. Es lassen sich Klassen mit gemeinsamen Merkmalen zu noch allgemeineren Klassen (abstrakten Basisklassen) zusammenfassen. Dabei werden allgemeine, also den ausgewählten Klassen gemeinsame Merkmale, in der übergeordneten Klasse zusammengefasst. Es entstehen dadurch sogenannte Klassenhierarchien. Auch dieses Prinzip ist uns in unserem Denken und in unserer Sprache allgegenwärtig. Es dient der Zeitersparnis. Allgemeines muss nicht jedes Mal wiederholt werden, es reicht die Einordnung in eine Überkategorie.

Für Klassen lassen sich zusätzlich folgende Thesen aufstellen:

- Klassen mit gemeinsamen Merkmalen können zu Basisklassen (Superklassen) zusammengefasst werden
- abgeleitete Klassen erben die Merkmale ihrer Basisklassen
- Eine Klasse kann mehreren Basisklassen angehören

Die Wiederverwendung von allgemeinen Merkmalen ist in unserem Denken und in unserer Sprache und somit auch in der Objektorientierung sehr effizient und elegant gelöst. Dieses Konzept stellt einen ernst zu nehmenden wirtschaftlichen Aspekt dar und hat der Objektorientierung letztendlich über deren Mächtigkeit bei der Wiederverwendung zum Siegeszug verholfen. Klassenbibliotheken und Frameworks verstärken die angesprochenen Effekte.

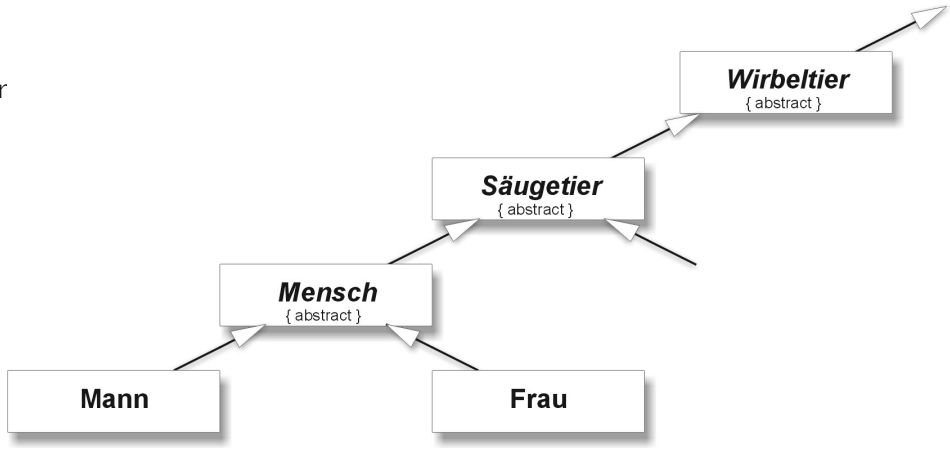


Abbildung 2: Hierarchie von Klassen

Kapselung

lat. capsula „Kästchen" von capsa „Behältnis"
Bedeutung: kleines Behältnis, Schutzhülle um Pflanzensamen, vgl. Sichtbarkeit, information hiding, Datenabstraktion
engl. capsule

Wenn wir uns die Dinge auf dieser Welt näher anschauen bemerken wir, dass vieles von der Natur geschaffene über eine erstaunliche Stabilität und Anpassungsfähigkeit verfügt. Selbst bei Konfrontationen mit anderen Objekten sind die existenziellen Merkmale der Kontrahenten oft gut geschützt.

Der wichtigste Schutzmechanismus, den die Natur entwickelt hat, ist das Verbergen des schützenswürdigen Merkmals. Wichtige Organe liegen bei Lebewesen oft weiter im Inneren (das Herz, die Lungen), sind von außen nicht sichtbar oder gar von festeren Strukturen, wie Knochen, geschützt (das Gehirn). Das Verbergen ist nicht nur ein Schutzmechanismus der Strukturmerkmale, das kann auch auf Verhaltensmerkmale zutreffen. So verbergen wir als Menschen gegenüber unseren Nächsten oft unsere wahren Gedanken. Das schützt uns im Alltag enorm. Wir setzen gezielt eine ganze Hierarchie von Schutzstufen für das was wir denken ein. Es gibt Dinge, die dürfen alle wissen, dann wiederum sagen wir manches nur im Familienkreis und es gibt Geheimnisse, die nehmen wir mit ins Grab. Die geschützten Merkmale eines Objektes sind also von außen für andere Objekte nicht sichtbar. Der Fachbegriff für diese Art Schutz lautet „Sichtbarkeit".

In der Objektorientierung unterscheidet man derzeit folgende Sichtbarkeiten:

- public, ein für alle Objekte im System sichtbares Merkmal
- package, ein für alle Objekte im selben Raum sichtbares Merkmal
- protected, sichtbar für alle Objekte dieser Klasse und deren Ableitungen
- private, nur sichtbar für alle Objekte dieser Klasse

Dieses Prinzip wird angewendet, um stabile Objekte herzustellen. Dabei können über die Klassenbeschreibung den einzelnen Merkmalen differenzierte Sichtbarkeiten zugeordnet werden, die sich wie Schalen um die zu schützenden Merkmale eines Objektes legen. Mit diesem Konzept kann durch den Entwickler ein klares System von Zugriffsrechten innerhalb einer Anwendung aufgebaut werden. Das ermöglicht es, vor allem in der verteilten Entwicklung von Systemen, Stabilität sicherzustellen.

Nachricht

frühneuhochdeutsch nachrichtung „das, wonach man sich ausrichten kann/soll“
Bedeutung: Mitteilung, Botschaft vgl.: Aufforderung, Befehl, Anweisung
engl. message, notification

Das Prinzip der Autonomität von Objekten und der Kapselung von Merkmalen wird durch das Konzept der Nachricht vervollständigt. Es unterscheidet sich signifikant vom Konzept des Befehls (vgl. imperative Programmierung). Zum Beispiel muss bei der Ausführung eines Befehls eine bestimmte Eingabe zu einer determinierten Ausgabe führen. Das Element, welchem der Befehl erteilt wurde, hat keine Entscheidungshoheit über das auszuführende Verhalten. Die Entscheidung ist bereits vor dem Aufruf des Befehls erfolgt. Bei einer Nachricht jedoch hat der Empfänger die Entscheidungshoheit über das konkret auszuführende Verhalten. Der Sender einer Nachricht entscheidet nicht über das Verhalten, sondern ist „nur noch“ für die Zustellung an den richtigen Empfänger zuständig. Es kann also bei dieser Art ein System zu denken und zu bauen, durchaus sein, dass einzelne Ausprägungen gar nicht oder nicht wie erwartet auf Nachrichten reagieren. Manchem Programmierer stehen jetzt die Haare zu berge. Aber gehen wir das gelassener an. Denken wir an unsere nähere deutsche Vergangenheit. Das System DDR, mit dem diktatorischen Konstruktionskonzept einer strengen befehlsorientierten Hierarchie, versteinerte und kollabierte, als Befehle nicht mehr befolgt wurden. Das Subsidiaritätsprinzip, als konstruktives Merkmal unserer Demokratie, verträgt eine Menge unangepasstes Verhalten ohne zu kollabieren. Im Gegenteil, die Summe des individuellen eigenverantwortlichen Handelns macht das gesamte System letztlich erst stabil und ermöglicht diesem seine ständige Weiterentwicklung. Derartige Flexibilität, Anpassungsfähigkeit bei gleichzeitiger Stabilität, ist ganz gewiss auch ein Ziel jedes Programmierers. Doch dazu mehr beim Thema Polymorphie.

Für das Konzept der Nachricht lassen sich erst einmal folgende Thesen formulieren:

- Objekte interagieren über Nachrichten
- Eine Nachricht hat einen Sender und einen Empfänger
- Bei der Übermittlung tritt ein Sendeereignis, gefolgt von einem Empfangsereignis auf, die Übermittlung benötigt Zeit
- Über Nachrichten können Merkmale verändert werden und Nachrichten können eine Antwort zur Folge haben
- Der Empfänger entscheidet selbst über das konkrete Verhalten als Reaktion auf die Nachricht
- Der Sender kann, aber muss nicht, auf eine Antwort warten
- Sendeereignis, Empfangsereignis und mögliche Antwort erfolgen nacheinander und nicht gleichzeitig

Und mal ehrlich, für unser Leben, so kompliziert und widersprüchlich es sein mag, favorisieren wir ein eigenverantwortliches Konzept, welches die Freiheit zur eigenen Entscheidung einbezieht und halten ein befehlsorientiertes System für eher unterentwickelt. Nur in der Programmierung können wir oft nicht so weit denken.

Polymorphie

altgriechisch πολυμορφία (Polymorphia) Vielgestaltigkeit
Bedeutung: veränderliche Merkmale bei gleicher innerer Zusammensetzung
engl. polymorphic, polymorphy, polymorphism

Polymorphie steht für Vielgestaltigkeit. Dieses Konzept zielt darauf ab, dass Mitglieder einer bestimmten Kategorie, also Objekte einer Klasse, sich nicht zwangsläufig stereotyp verhalten müssen. Wir haben gerade besprochen, dass in der Objektorientierung Entscheidungen dezentral, also bei

den Objekten selbst erfolgen. Jedes Objekt bringt seine Logik selbst mit und entscheidet für sich selbst. Die zentrale Programmlogik ist nicht mehr der Befehlsgeber, sondern der Moderator.

Polymorphie beschreibt folgenden Effekt: Wenn an eine Menge von Ausprägungen (Objekte) einer Klasse ein und dieselbe Nachricht gesendet wird, reagieren diese mit sichtbar oder auch unsichtbar unterschiedlichem Verhalten. Wenn also der Dozent allen Instanzen vom Typ „Student" die Nachricht sendet: „... lernt die Basiskonzepte bis morgen auswendig...", wird jeder Student ein sehr individuelles Verhalten an den Tag legen. Manche werden sehr fleißig lernen, manche ein bisschen und manche gar nicht. Der Sender der Nachricht hat nur mittelbaren Einfluss auf das Verhalten. Im ersten Augenblick erscheint uns das für die Programmierung wenig hilfreich, aber diese recht lose und freie Form der Zusammenarbeit in einem System hat auch Vorteile. Lernt ein Student nicht, geht der Unterricht am nächsten Tag trotzdem weiter. Das gesamte System selbst funktioniert zur Laufzeit stabil, auch wenn einzelne Instanzen manchmal nicht wie erwartet reagieren. Objektorientierte Sprachen bieten für die Realisierung von Polymorphie entsprechende Ausdrucksmittel. Das Interface-Konzept in JAVA oder auch C# ist konsequent polymorph. Immer dann, wenn in objektorientierten Sprachen vom Überschreiben oder Überladen von Methoden die Rede ist, handelt es sich um Konstruktionsmittel für dynamische oder statische Polymorphie. Dass moderne Anwendungen Funktionalität und Aussehen soweit wandeln können, dass man diese kaum noch wiedererkennt, erreicht man über sogenannte Plug-Ins, Skins usw. Dabei werden diese Komponenten von Entwicklern programmiert, die eigentlich nur die Schnittstelle (gemeinsame Nachrichtensignatur) zum System kennen, in welches sie sich einfügen. Dies alles basiert auf Polymorphie, die ermöglicht, dass sich durch Kombination unterschiedlicher Objekte zur Laufzeit ein gewünschtes Verhalten einstellt. Ohne Umprogrammieren, Kompilieren und Linken der Anwendung bleibt das gesamte System trotzdem stabil. Polymorphie ist sozusagen die hohe Schule der Objektorienteierung.

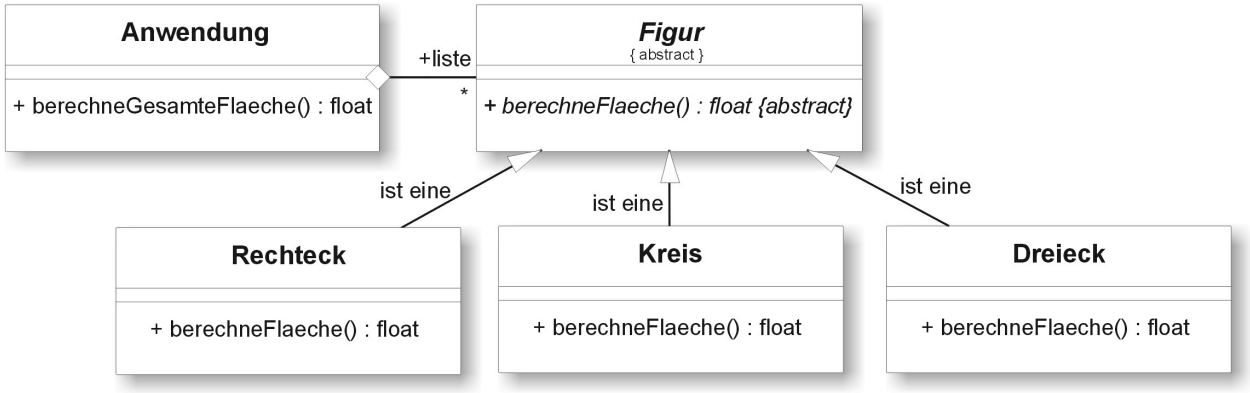


Abbildung 3: *Polymorphie*

Auch ohne UML Kenntnisse zeigt dieses Klassendiagramm dem Betrachter einen typischen Anwendungsfall für Polymorphie. Die Anwendung muss zur Ermittlung der Gesamtfläche die Liste der Figuren in einer Schleife durchlaufen. Dabei sendet sie jeder Figur die Nachricht ***berechneFlaeche()*** und kumuliert die Antworten zur Gesamtfläche. Die Liste selbst enthält als Instanzen Rechtecke, Kreise und Dreiecke, die jeweils ihre eigene Berechnung der Fläche mitbringen. Die Anwendung selber muss keine Unterscheidung hinsichtlich der Figur vornehmen. Selbst wenn eine neue Figur, zum Beispiel eine Ellipse, hinzugefügt wird, sind weder an der Klasse Figur noch an der Klasse Anwendung Änderungen nötig. Zur Laufzeit wird durch das Überschreiben der Methode ***berechneFlaeche()*** in der Ellipse automatisch der korrekte Flächeninhalt einer Ellipse berechnet.

Zum Ende meiner Einlassung möchte ich die verbleibenden Konzepte noch kurz ansprechen um ein wenig die Vollständigkeit zu wahren.

Assoziation

lat. associare „vereinigen, verbinden, verknüpfen, vernetzen“
Bedeutung: miteinander in Beziehung stehen, vgl. Relationship
engl. association, relationship

Systeme bestehen aus Objekten, die zusammenarbeiten. Objekte interagieren über den Austausch von Nachrichten. Dafür müssen diese sich kennen. Diesen Umstand nennt man Assoziation. Die Objekte eines Systems stehen in Beziehung zueinander. In der konkreten Programmierung heißt das man muss den Instanznamen für das Objekt kennen, um ihm Nachrichten senden zu können. Bei der Assoziation kennen sich die Objekte, aber sind nicht füreinander verantwortlich.

Aggregation

l*at. aggregatio,* „Anhäufung, Vereinigung“
Bedeutung: Ganz-Teil-Beziehung, Das Ganze setzt sich aus Teilen zusammen
engl. aggregation

Systeme bestehen aus Objekten. Das System selbst ist auch ein Objekt und es ist sinnvoll, komplexe Systemteile als Komponenten zusammenzufassen, die wiederum Objekte sind, welche aus Objekten bestehen. Derartige Ganz-Teil-Beziehungen (part of) sind eine wesentliche Grundlage für die zweckmäßige Strukturierung von Systemen. Man bezeichnet diese als Aggregation. Dabei ist das Ganze jeweils für seine Teile verantwortlich. Die Aggregation ist eine Erweiterung des Konzeptes der Assoziation.

Komposition

lat. compositio „Zusammenstellung, Gestaltung“
Bedeutung: Ganz-Teil-Beziehung, Das Ganze setzt sich aus notwendigen Teilen zusammen bzw. die Teile benötigen das Ganze
engl. composition

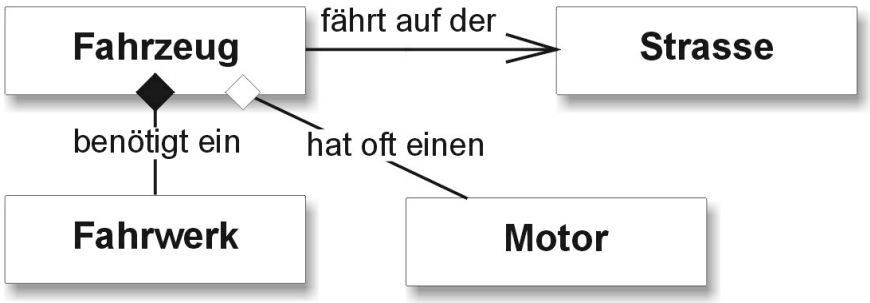


Abbildung 4: *Assoziation, Aggregation und Komposition*

Als Komposition bezeichnet man eine verschärfte Form der Aggregation. Bei einer Komposition stehen das Ganze und das Teil in einer existenziellen Abhängigkeit zueinander. Für dieses Konzept stehten in aktuellen Sprachen nur wenig Ausdrucksmöglichkeiten zur Verfügung.

Ein kurzes Fazit

Lange Rede kurzer Sinn. Unsere natürliche Sprache ist objektorientiert!

> Wenn der Taster gedrückt ist schalte die LED an.
> If the button is pressed the LED will turn on.
> if button.isPressed then led.on
> if (button.isPressed()) led.on();

Zum Autor, und seinen Produkten

Technisches Umfeld:

Seit 1993 ist Alexander Huwaldt nebenberuflich als Dozent an der Berufsakademie Bautzen, Dresden und an weiteren Hochschulen und Universitäten tätig in den Bereichen objektorientierter Systementwurf, integrierte Informationssysteme, Geschäftsprozess-modellierung, Rechnerarchitektur, Einführung in die Informatik, Software Engineering,

Desweiteren als UML Trainer unter anderem bei Lufthansa Systems, Deutsche Post.

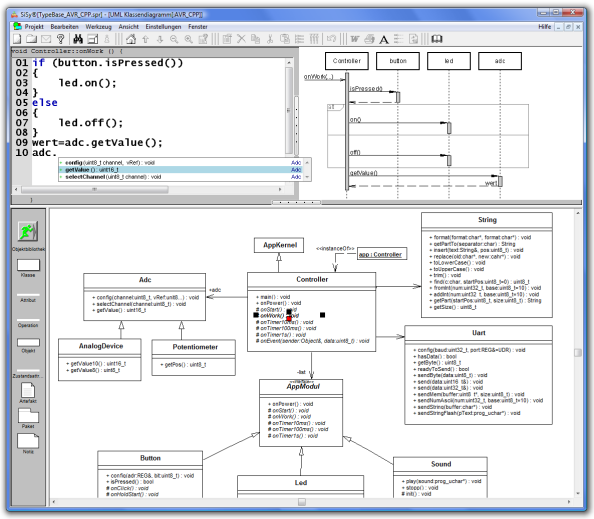
2005 Hat er die im Rahmen des OMG Certification Program die Ausbildung zum OMG Certified UML Professional absolviert.

myAVR®:

Das Mikrocontroller-Lernsystem myAVR bietet auf der Basis der 8 Bit AVR Mikrocontrollerserie von Atmel Komplettsysteme als Lernmittel von der Berufsausbildung bis zum Universitätsstudium an. Für die didaktische Gestaltung der Systeme ist der Autor maßgeblich verantwortlich. myAVR ist ein eingetragenes Warenzeichen der Laser & Co. Solutions GmbH *(www.myAVR.de)*.

SiSy®:

Simple System ist als Projekt sozusagen das Baby des Autors. Technisch gesehen handelt es sich bei SiSy um ein Modellierungswerkzeug auf der Basis einer Metamodell-Engine. Über die ladbaren Metamodelle reicht das Einsatzspektrum des Tools von der Geschäftsprozessmodellierung bis zur Nutzung als CASE-Tool. Das wohl größte und spektakulärste Modell, das mit SiSy erstellt wurde, waren die betrieblichen Abläufe der EXPO2000 in Hannover. Über den integrierten Dokumentationsgenerator konnten Pläne und Darlegungen der Arbeitsabläufe für über 20.000 EXPO Mitarbeiter vollständig aus dem Modell heraus konsistent generiert werden. SiSy ist ein eingetragenes Warenzeichen der Laser & Co. Solutions GmbH (www.SiSy.de).



SiSy AVR ist eine spezielle Ausgabe für die Entwicklung von eingebetteten Systemen mit AVR-Controllern. Dabei verfügt das Tool über alle wesentlichen Fähigkeiten einer kompletten Entwicklungsumgebung und ermöglicht mit Codegeneratoren aus UML-Modellen ein echtes Round Trip Engineering.

SiSy ARM heißt das neueste Mitglied in der SiSy Familie und wird in Zusammenarbeit mit Avnet SILICA als Bundel mit dem STM32F4 Discovery Kit angeboten.

Andreas Willert live zu den Themen:

- **Funktionelle Sicherheit / DIN EN 61508**
  Entwicklung von Hardware für sicherheitsgerichtete Anwendungen
  11. + 12.07.2012 Design & Elektronik Entwicklerforum in München
- **Anforderungsmanagement Workshop**
  12.07.2012 Design & Elektronik Entwicklerforum in München
- **Neuer Core, neues Game...**
  11. + 12.07.2012 Konferenz für ARM Systementwicklungen, München
- **Software Architekturdesign, Voraussetzungen für die wichtigsten Qualitätsattribute**
  04. - 06.12.2012 ESE - Kongress in Sindelfingen

Trainings und Schulungen bei Willert:

EMBEDDED UML START-UP TRAININGS

1. Hands-on exercises based on Rational® Rhapsody® in C
2. Hands-on exercises based on SPARX Systems® Enterprise Architect

REQUIREMENT ENGINEERING START-UP TRAININGS

1. Hands-on exercises based on Rational® DOORS®
2. Hands-on exercises based on Polarion®

SOFTWARE ARCHITEKTUR-DESIGN

für Embedded Systeme / Workshop

Weitere Ausgaben des

EMBEDDED SOFTWARE ENGINEERING REPORT

finden sie unter:

http://www.willert.de/Newsletter

Autor:

DIPL. ING. PÄD. ALEXANDER HUWALDT

Herausgeber:

WILLERT SOFTWARE TOOLS GMBH

Hannoversche Straße 21
31675 Bückeburg
www.willert.de
info@willert.de
Tel.: +49 5722 9678 - 60