

Implementierung

Objektorientierung

Objektorientierte Programmierung mit C

08.03.2010 | Autor: Frank Listing*

Obwohl C keine objektorientierte Sprache ist, ist die objektorientierte Programmierung mit ihr möglich. Dieser Artikel beschäftigt sich mit der Umsetzung von UML-Elementen aus dem Klassendiagramm in C-Code. Es wird gezeigt, was möglich ist und wo die Umsetzung mit der Programmiersprache C an ihre Grenzen stößt.



*Dipl.-Ing. Frank Listing ist seit 2002 Trainer und Projektcoach bei der MicroConsult GmbH mit dem Schwerpunkt Microsoft-Plattformen, objektorientierte Programmierung und Testen von Embedded Systemen und u.a. fachlich für das Thema .NET verantwortlich.

Die UML (Unified Modeling Language) hat sich mittlerweile auch im Embedded-Bereich als eine akzeptierte Notation etabliert. Sie wird in immer mehr Projekten eingesetzt, um den Aufbau und das Verhalten der Programme zu beschreiben.

Dabei gibt es verschiedene Bereiche in der UML, die mehr oder weniger nah am Code sind. Wo ein Paketdiagramm noch die Architektur beschreibt und weitgehend unabhängig von der später verwendeten Programmiersprache ist, muss bei der Erstellung von Klassendiagrammen schon an die spätere Implementierung gedacht werden, da nicht jedes in der UML unterstützte Merkmal auch in jeder Programmiersprache verfügbar ist.

Die bei der Embedded-Programmierung zurzeit am häufigsten verwendete Programmiersprache C ist nicht objektorientiert. Dies dient in vielen Projekten als Ausrede, um nicht über neue Methoden der Programmierung nachdenken zu müssen. Dabei findet objektorientierte Entwicklung im Kopf des Entwicklers statt. Sie lässt sich mit jeder Programmiersprache

umsetzen.

Bei einer nicht objektorientierten Programmiersprache muss mit zusätzlichen Regeln und Techniken ein Umfeld geschaffen werden, das diese Art der Programmierung ermöglicht.

UML-Klassen in C umsetzen



Eines der wichtigsten Elemente in der UML ist die Klasse. Sie steht im Mittelpunkt der objektorientierten Programmierung. Das Klassendiagramm der UML stellt die im Projekt verwendeten Klassen mit ihren Attributen (entspricht den Daten) und Operationen (entspricht den Funktionen) und andererseits die Beziehungen zwischen den Klassen dar.

In der Programmiersprache C gibt es kein Syntaxelement für die Klasse. Allerdings gibt es die Struktur als Mittel, komplexe Daten strukturiert abzulegen. Im einfachsten Fall wird eine Struktur mit einem zugeordneten Satz von Funktionen definiert. Per Programmervorschrift wird festgelegt, dass auf die Struktur nur über die speziellen Funktionen zugegriffen werden darf.

Listing 1-1, eine Klasse in C:

```
typedef struct
```

```

{
int anzahlRaeder;
char* hupTon;
int geschwindigkeit;
}Auto;

void Auto_init(Auto *this, int anzahlRaeder, char* hupTon);
void Auto_fahre(Auto *this);
void Auto_hupe(Auto *this, int anzahl);

```

Zuordnung von Funktionen und Struktur

Die Zuordnung von Funktion und Struktur erfolgt über den ersten Parameter der Funktion – das ist immer ein Zeiger auf die Instanz der Struktur, die gerade bearbeitet wird. Dieser Parameter entspricht dem this-Pointer in C++.

Listing 1-2, Objekterzeugung auf dem Stack:

```

// Objekt auf dem Stack

Auto a;

Auto_init(&a, 4, „tut“);

Auto_fahre(&a);

Auto_hupe(&a, 3);

```

Listing 1-3, Objekterzeugung auf dem Heap:

```

// Objekt auf dem Heap

Auto *pa= (Auto*)malloc(sizeof(Auto));

Auto_init(pa, 3, „maep“);

Auto_fahre(pa);

Auto_hupe(pa, 2);

free(pa);

```

Das Problem dieser einfachen Lösung ist, dass gegen Vorschriften auch gern mal verstoßen wird und damit das objektorientierte Gebäude zum Einsturz gebracht wird. Auf dieses Dilemma wird später noch näher eingegangen.

Beziehungen zwischen den Klassen implementieren



Nach den Klassen geht es an die Darstellung der Beziehungen zwischen ihnen. Im Klassendiagramm werden Assoziation, Aggregation und Vererbung verwendet. Die Assoziation ist eine einfache Beziehung zwischen zwei gleichrangigen Klassen (Bild 2).

Die Umsetzung im C-Code ist einfach: Eine Klasse enthält einen Zeiger auf die andere Klasse. Bei der gerichteten Assoziation (siehe Abbildung) hat nur eine von beiden Klassen einen Zeiger auf die andere, und bei einer bidirektionalen Assoziation haben beide Klassen einen Zeiger auf die jeweilige andere Klasse.

Listing 2-1:, die Klasse Garage enthält einen Zeiger auf die Klasse Auto:

```
typedef struct
{
    Auto* pAuto;
}Garage;

void Garage_init(Garage *this);
void Garage_init2(Garage *this, Auto *pAuto);
void Garage_einparken(Garage *this, Auto *pAuto);
void Garage_ausparken(Garage *this);
```

Da die einzelnen Objekte erst einmal nichts voneinander wissen, muss im Programmcode noch eine Verbindung zwischen ihnen hergestellt werden:

Listing 2-2: In der Funktion Garage_einparken() wird die Assoziation initialisiert.

```
Auto a;

Garage g;

Auto_init(&a, 4, „tut“);

Garage_init(&g);

Garage_einparken(&g, &a);
```

Umsetzung von Aggregation in C



Die Aggregation ist eine „besteht aus“-Beziehung, hier wird ein Objekt in ein anderes eingebettet (Bild 3). Eine Aggregation kann realisiert werden, indem das einzubettende Objekt als Member in die Struktur des äußeren Objektes aufgenommen wird.

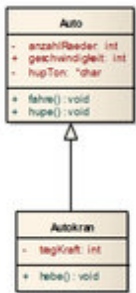
Listing 3, die Klasse Auto aggregiert ein Objekt der Klasse Motor:

```
typedef struct
{
    int anzahlRaeder;
    char* hupTon;
    int geschwindigkeit;
    Motor motor;
```

```
}Auto;
```

Genau genommen ist diese Beziehung sogar eine Komposition, die strengere Form der Aggregation, da das eingebettete Objekt mit dem äußeren Objekt zerstört wird.

Umsetzung von Vererbung in C



Bei der Vererbung werden Daten und Funktionen einer Basisklasse in einer abgeleiteten Klasse wiederverwendet und erweitert (Bild 4). In der Programmiersprache C kann tatsächlich auch eine Vererbung realisiert werden. Da der ANSI-Standard die Anordnung der Daten im Speicher regelt, kann durch Aggregation der Basisklasse als erstes Element in der Struktur der abgeleiteten Klasse eine Vererbung erreicht werden.

Listing 4-1, eine Basisklasse:

```
typedef struct
{
    int anzahlRaeder;
    char* hupTon;
    int geschwindigkeit;
    Motor motor;
}Auto;

void Auto_init(Auto *this, int anzahlRaeder, char* hupTon);
void Auto_fahre(Auto *this);
void Auto_hupe(Auto *this, int anzahl);
void Auto_setGeschwindigkeit(Auto *this, int geschwindigkeit);
int Auto_getGeschwindigkeit(Auto *this);
```

Listing 4-2, eine abgeleitete Klasse:

```
typedef struct
{
    Auto;
    int tragKraft;
}Autokran;

void Autokran_init(Autokran *this, int tragKraft);

void Autokran_init2(Autokran *this, int tragKraft, int anzahlRaeder, char*
hupTon);
```

```
void Autokran_hebe(Autokran *this, int gewicht);
```

Listing 4-3, die abgeleitete Klasse kann die Funktionen der Basisklasse benutzen:

```
void test(void)
{
    Autokran kran;
    Autokran_init(&kran, 50);
    Auto_setGeschwindigkeit((Auto*)&kran, 20);
    Auto_hupe((Auto*)&kran, 3);
    Auto_setGeschwindigkeit((Auto*)&kran, 0);
    Autokran_hebe(&kran, 20);
    Autokran_hebe(&kran, 60);
}
```

Durch das Einfügen der Basisklassendaten als erstes Element in die Struktur werden diese auch als erstes in den Speicher gelegt. Wird dann eine Funktion der Basisklasse aufgerufen, findet diese die ihr bekannten Daten.

Daten vor unberechtigter Manipulation schützen

Bisher wurde die Datenstruktur der Klasse öffentlich im Header abgelegt, und es wurde davon ausgegangen, dass sich jeder Entwickler brav an die Vorschriften hält und auf die Struktur nie direkt, sondern nur über die zugeordneten Funktionen zugreift. Die Realität kennt aber nicht nur brave Entwickler. Sei es Unwissenheit oder Bequemlichkeit, es wird nicht lange dauern, bis jemand direkt auf die Daten zugreift. Die nächste Fehlersuche wird dann zur Odyssee.



Objektorientierte Programmiersprachen wie C++ bieten zum Schutz der Klassenelemente spezielle Schlüsselwörter an, und der Compiler prüft, ob der Zugriff berechtigt ist. Das gibt es in C leider nicht.

Allerdings kann mit kleinen Änderungen am bisherigen Code trotzdem ein Schutz von internen Daten und Funktionen erreicht werden. Es können nicht alle bekannten Schutzstufen umgesetzt werden, aber zwischen öffentlichen (public) und privaten (private) Klassenelementen kann unterschieden werden. Voraussetzung dafür ist, dass die der Klasse zugrundeliegende Struktur nicht mehr veröffentlicht wird (Bild 5).



Die Struktur der Klasse wird nicht mehr öffentlich bekanntgegeben. Es werden lediglich Funktionen zum Erzeugen eines Objekts und zum Arbeiten mit dem Objekt veröffentlicht. Die Struktur der Klasse und die Implementierung der Funktionen werden nur als Bibliothek (Lib) verteilt und sind damit für den Benutzer der Klasse nicht mehr sichtbar. Neben dem Vorteil, dass nicht mehr auf private Elemente zugegriffen werden kann, ist es jetzt sogar möglich, das Objekt, ähnlich einem Konstruktor im C++, bei der Erzeugung zu initialisieren.

Abschließend kann gesagt werden, dass in der Programmiersprache C deutlich mehr objektorientierte Elemente aus der UML umgesetzt werden können, als auf den ersten Blick ersichtlich ist. Sogar Schutzmechanismen für private Daten sind möglich. Damit können auch C-Programme übersichtlicher und sicherer gestaltet werden.

Redakteur: Martina Hafner

Die Beiträge auf dieser Website sind urheberrechtlich geschützt. Bei Fragen zu den Nutzungsrechten wenden Sie sich bitte an manuela.maurer@vogel.de oder Tel.: 0931-418-2888.

Dieses PDF wurde Ihnen bereitgestellt von <http://www.elektronikpraxis.vogel.de>

Bildergalerie

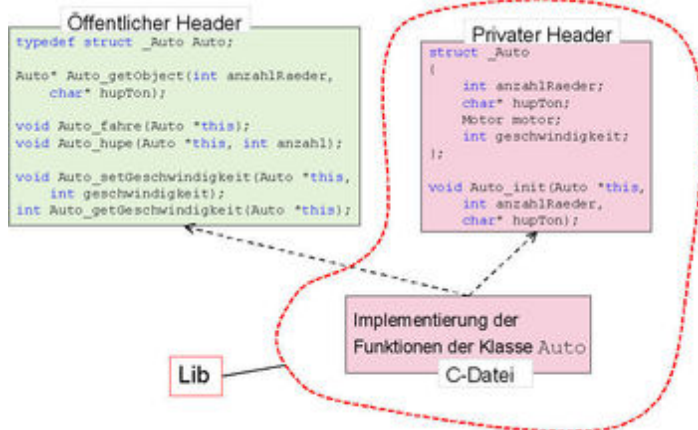


Bild 6: Das Programm sieht nur noch die öffentlichen Funktionen.

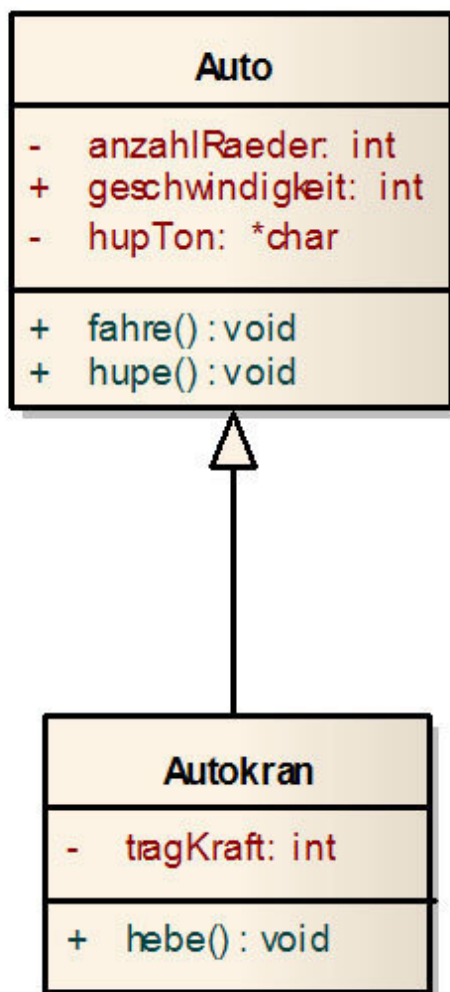


Bild 4: UML-Diagramm für die Vererbung. Die Basisklasse Auto vererbt ihre Daten und Funktionen an die abgeleitete Klasse Autokran weiter, die wiederum erweitert werden kann (z.B. um das Attribut `tragKraft` vom Typ `integer`). In der Programmiersprache C kann tatsächlich auch eine Vererbung realisiert werden.



Bild 3: Die Aggregation der UML ist eine „besteht aus“-Beziehung. Das Auto besteht (unter anderem) aus einem Motor. Eine Aggregation kann in C realisiert werden, indem das einzubettende Objekt als Member in die Struktur des äußeren Objektes aufgenommen wird.

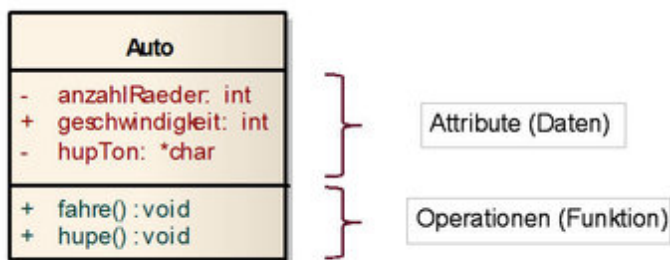


Bild 1: Eine Klasse in UML dargestellt. Die Klassen enthalten Attribute (Daten, z.B. Attribut geschwindigkeit vom Typ integer) sowie Operationen (Funktionen, z.B. `fahre():void`).



Bild 2: UML-Diagramm einer gerichteten Assoziation. Das Objekt namens Garage steht in Beziehung zum Objekt Auto. Solche Beziehungen werden in C durch Zeiger umgesetzt.

