Objektiv Denken III: Die Beinhaltung in MATLAB

Ernesto Rico-Schmidt

Kurzfassung— **Im ersten Teil dieser Serie wurde als Einstieg in die objektorientierte Programmierung in Matlab ein einfaches Beispiel besprochen. Die Realisierung einfacher objektorientierten Konzepte in Matlab wurde Anhand eines einfachen Beispiels erläutert.**

Im zweiten Teil wurde dann die Realisierung der Datenkapselung, der Zugriffsmethoden (`get` und `set`), sowie der einfachen Vererbung besprochen. Ein Wasserversorgungssystem, bestehend aus Quellen, Senken, Tanks, Ventile, und Leitungen wurde modelliert.

In diesem dritten und letzten Teil der Serie wird ein weiteres wichtiges Konzept der objektorientierten Programmierung behandelt: Die Beinhaltung. Die einzelnen Komponenten werden zu einem System zusammengefaßt, um dieses anschließend zu simulieren.

I. Beinhaltung in Matlab

UNTER Beinhaltung (engl. *aggregation*) versteht man das Zusammenfassen von mehreren Objekten zu einem. So kann eine weitere Ebene der Abstraktion mit den bereits besprochenen Vorteilen erreicht werden.

Die Realisierung der Beinhaltung in MATLAB ist etwas kompliziert, denn man kennt hier keine Referenzen (*pointer*), man muß also die Objekte in eine Liste zusammenfassen. Objekte (oder Elemente) von verschiedenen Typen können nur mit Hilfe von *cells* zusammengefaßt werden. Diese sind mehrdimensionale *arrays*.

II. Die Klasse system

Die Komponente, die im zweiten Teil dieser Serie besprochen wurden, werden nun in einem System zusammengefaßt. Die Klasse `system` enthält den Namen und die Größe des System (Anzahl der Komponente) sowie die Komponenten selbst.

A. Der Konstruktor

Der Konstruktor ermöglicht drei Aufrufe, die Unterscheidung zwischen den Aufrufe erfolgt über die Anzahl der Eingangsparameter, `nargin`.

- *Kein Argument* Erhält der Konstruktor kein Argument, dann wird ein *default*-Objekt erzeugt.
- *Objekt als Argument* Ist das Argument im Aufruf ein Objekt derselben Klasse `system`, dann wird einfach eine Kopie des Objekts zurückgegeben.
- *Komponente als Argumente* Werden Argumente übergeben, die kein Objekt der Klasse `system` sind, dann wird Anhand dieser ein neues Objekt (ein System) erzeugt. Die Komponente werden zunächst einfach hintereinander angefügt und die `connect`-Methoden aufgerufen um die Parameter `inflowPress`, `inflowRes`, `outflowPress` und `outflowRes` anzugleichen um die Simulation zu ermöglichen.

@system/system.m

```
function s = system(varargin)
% SYSTEM class constructor

switch nargin
 case 0         % default object
  s.name = 'none';
  s = class(s, 'system');
 case 1         % copy object
  if (isa(name, 'system')),
    c = name;
  else
    error('wrong argument type');
  end
 otherwise        % new object
  s.name = varargin{1};
  s.length = length(varargin)-1;
  for i = 1:s.length,
    s.components(i) = {varargin{i+1}};
  end
  for j = 1:s.length-1,
    [s.components{j}, s.components{j+1}] = ...
      connect(s.components{j}, s.components{j+1});
  end
  s = class(s, 'system');
end
```

B. Die `get`- und `set`-Methoden

Für die Simulation benötigt man Zugriff auf die einzelnen Objekte des Systems. Man muß die aktuellen Werte der Komponenten abfragen können, bzw. diese wieder nach jedem Simulationsschritt aktualisieren, siehe dazu die `update`-Methoden.

@system/get.m

```
function c = get(s, i)
% GET method for SYSTEM class

if i <= s.length
  c = s.components{i};
else
  error([num2str(i) 'th component is not defined']);
end
```

Die `set`-Methode hat die Besonderheit, daß das veränderte Objekt `c` als Parameter übergeben wird, die Methode wird dann ein verändertes Objekt (das System) zurückgeben, das dann neu zugewiesen werden muß:

```
>> s = set(s, 2, c)
```

Hier wird die veränderte Komponente `c` an der zweiten Stelle des Systems `s` neu gesetzt.

@system/set.m

```
function s = set(s, i, c)
% SET method for SYSTEM class

if i <= s.length
  s.components{i} = c;
else
  error([num2str(i) 'th component is not defined']);
end
```

C. Die *simulate*-Methode

Da das Verhalten der einzelnen Komponenten unterschiedlich ist, wird die eigentliche Simulation nicht im der Klasse system implementiert, sondern durch die update-Methoden der Komponenten. Die Methode simulate steuert bzw. delegiert die Simulation an die Objekte des Systems.

@system/simulate.m

```
function simulate(s, n)
% SIMULATE method for SYSTEM class

for i = 1:n,
  for j = 1:s.length-1,
    [temp1, temp2] = update(get(s, j), get(s, j+1));
    s = set(s, j, temp1);
    s = set(s, j+1, temp2);
  end
  temp3 = update(get(s, s.length));
  s = set(s, s.length, temp3)
end
```

III. Die Verbindung der Komponenten

Jede Komponente des Systems ist durch eine Reihe von Attributen gekennzeichnet, die den Zustand zu einem bestimmten Zeitpunkt beschreiben.

- `inflow`, `outflow` Zu- und Abfluß, die Menge Wasser, die pro Zeiteinheit (`tick`) aus bzw. in die Komponente fließt.
- `inflowPress`, `outflowPress` Zu- und Abflußdruck[1], die Menge Wasser, die in der nächsten Zeiteinheit in die bzw. aus der Komponente fließen würde.
- `inflowRes`, `outflowRes` Zu- und Abflußgegendruck, die Menge Wasser, die eine Komponente bzw. die nächste in der nächsten Zeiteinheit verträgt oder verlangt.

A. Die *connect*-Methoden

Um die Simulation erst möglich zu machen, müssen die Attribute der nacheinander liegenden Komponenten angeglichen werden. So "weiß" jede Komponente von der Existenz der nächsten.

Wenn die Komponente `c` mit `cc` verbunden wird, dann müssen der Zuflußdruck von `cc` gleich dem Abflußdruck von `c` und der Abflußgegendruck von `c` gleich dem Zuflußgegendruck von `cc` gesetzt werden.

[1]Der Ausdruck "Druck" ist in diesem Zusammenhang leider verwirrend.

@component/connect.m

```
function [c, cc] = connect(c, cc)
% CONNECT method for COMPONENT class

cc = set(cc, 'inflowPress', get(c, 'outflowPress'));
c = set(c, 'outflowRes', get(cc, 'inflowRes'));
```

Eine Leitung kann eine maximale Menge Wasser (`maxFlow`) führen. Dieser Fall ist bei der Verbindung der Komponenten (und auch bei der Simulation) gesondert zu behandeln. Die restlichen Attribute werden die Methode der Basis-Klasse gesetzt.

@pipe/connect.m

```
function [p, c] = connect(p, c)
% CONNECT method for PIPE class

p = set(p, 'inflowRes', ...
      min(get(p, 'maxFlow'), get(c, 'inflowRes')));

[p.component, c] = connect(p.component, c);
```

Ein Ventil kann entweder offen oder geschlossen sein, dies wird auch die Weitergabe der Attribute beeinflussen. Wiederum werden die restlichen Attribute durch die Methode der Basis-Klasse gesetzt.

@valve/connect.m

```
function [v, c] = connect(v, c)
% CONNECT method for VALVE class

if get(v, 'status'),
  v = set(v, 'inflowRes', get(c, 'inflowRes'));
else
  v = set(v, 'inflowRes', 0);
end

[v.component, c] = connect(v.component, c);
```

IV. Die Simulation der Komponenten

Die eigentlich Simulation wird durch die einzelnen Komponenten durchgeführt, da jede Komponente ihr eigenes Verhalten besitzt, das durch eigene update-Methoden implementiert wird. Die Methode der Basis-Klasse erledigt dabei den allgemeinen Teil der Simulation.

A. Die *update*-Methoden

Wenn Wasser von der Komponente `c` in `cc` fließt, dann müssen der Zufluß und der Zuflußdruck von `cc` gleich dem Abfluß bzw. dem Abflußdruck von `c`, sowie der Abflußgegendruck von `c` gleich dem Zuflußgegendruck von `cc` gesetzt werden.

@component/update.m

```
function [c, cc] = update(c, cc)
% UPDATE method for component

cc = set(cc, 'inflow', get(c, 'outflow'));
cc = set(cc, 'inflowPress', get(c, 'outflowPress'));
c = set(c, 'outflowRes', get(cc, 'inflowRes'));
```

Aus einem Tank kann eine maximale Menge Wasser (`maxOutflow`) fließen, daher werden der Abfluß und der

Abflußdruck das Minimum aus dem maximalen Abfluß und dem Abflußgegendruck sein.

Der Füllstand des Tanks wird neu aus Zu- und Abfluß berechnet bevor man die restlichen Attribute durch die Methode der Basis-Klasse aktualisiert.[2]

@tank/update.m

```
function [t, c] = update(t, c)
% UPDATE method for TANK class

t = set(t, 'outflowPress', ...
      min(get(t, 'outflowRes'), get(t, 'maxOutflow')));
t = set(t, 'outflow', get(t, 'outflowPress'));
t = set(t, 'inflow', get(t, 'inflowPress'));

delta = get(t, 'inflow') - get(t, 'outflow');
t = set(t, 'level', get(t, 'level') + delta);

[t.component, c] = update(t.component, c);
```

Das Verhalten der Leitung wird durch den maximalen Fluß `maxFlow` bestimmt. Das jeweilige Minimum zwischen Zuflußruck bzw. Abflußgegendruck und maximalen Fluß ergibt den Abflußdruck bzw. den Zuflußgegendruck. Der tatsächliche Fluß ist dann das Minimum zwischen Abflußdruck und Zuflußgegendruck.

@pipe/update.m

```
function [p, c] = update(p, c)
% UPDATE method for PIPE class

p = set(p, 'outflowPress', ...
      min(get(p, 'inflowPress'), get(p, 'maxFlow')));
p = set(p, 'inflowRes', ...
      min(get(p, 'outflowRes'), get(p, 'maxFlow')));
p = set(p, 'outflow', ...
      min(get(p, 'outflowPress'), get(p, 'inflowRes')));

[p.component, c] = update(p.component, c);
```

Ob ein Ventil entweder offen oder geschlossen ist wird die Simulation beeinflussen. Bei einem offenen Ventil werden die Attribute aktualisiert, ist dieses geschlossen werden sie einfach gleich Null gesetzt. Wiederum werden die restlichen Attribute durch die Methode der Basis-Klasse aktualisiert.

@valve/update.m

```
function [v, c] = update(v, c)
% UPDATE method for VALVE class

if get(v, 'status')
  v = set(v, 'outflowPress', get(v, 'inflowPress'));
  v = set(v, 'inflowRes', get(v, 'outflowRes'))
  v = set(v, 'outflow', ...
        min(get(v, 'outflowPress'), get(v, 'inflowRes'))):
else
  v = set(v, 'outflowPress', 0);
  v = set(v, 'inflowRes', 0);
  v = set(v, 'outflow', 0);
end

[v.component, c] = update(v.component, c);
```

Die Aktualisierung der Attribute der Senke ist besonders einfach, da sie am Ende des Systems ist und einfach das Minimum zwischen Zuflußdruck und -gegendruck in sie fließt.

@sink/update.m

```
function s = update(s)
% UPDATE method for SINK class

s = set(s, 'inflow', ...
      min(get(s, 'inflowPress'), get(s, 'inflowRes')));
```

Die Quelle ist ähnlich, aus dieser fließt einfach das Minimum zwischen Abflußdruck und -gegendruck bevor die restlichen Attribute aktualisiert werden.

@source/update.m

```
function [s, c] = update(s, c)
% UPDATE method for SOURCE class

s = set(s, 'outflow', ...
      min(get(s, 'outflowPress'), get(s, 'outflowRes')));

[s.component, c] = update(s.component, c);
```

B. Ein enfaches System

Ein einfaches System, bestehend aus einer Quelle `so`, einer Senke `si`, zwei Leitungen `p1`, `p2` und einem Tank `t`, kann so erzeugt werden.

```
>> clear all
>> clear classes
>> t = tank('t, 100, 100, 500, 600);
>> so = source('so, 300);
>> si = sink('si, 200);
>> p1 = pipe('p1, 150);
>> p2 = pipe('p2, 100);
>> s = system('sys', so, p1, t, p2, si);
```

Um das System dann für 5 Zeiteinheiten zu simulieren genügt der Befehl:

```
>> simulate(s, 5)
```

Die Ausgabe zeigt den Verlauf der Simulation, das Endergebnis:

```
      name: sys

     name: so (source)
  outflow: 150 l/tick

     name: p1 (pipe)
     flow: 150 l/tick

     name: t (tank)
   inflow: 150 l/tick
  outflow: 100 l/tick
    level: 250 l

     name: p2 (pipe)
     flow: 100 l/tick

     name: si (sink)
   inflow: 100 l/tick
```

[2]Die Attribute `maxLevel` und `minLevel`, werden der Einfachheit halber nicht berücksichtigt. Man könnte z.B. ein Alarm ausgeben, sobald Maximum oder Minimum erreicht werden.

V. Zusammenfassung

Die Realisierung der Beinhaltung in MATLAB wurde in diesem letzten Teil der Serie besprochen. Spätestens jetzt gelangt man zu dem Schluß, daß MATLAB die objektorientierte Programmierung *ermöglicht* aber nicht *unterstützt*.

> "A language is said to *support* a style of programming if it provides facilities that makes it convenient (reasonably easy, safe, and efficient) to use that style. A language does not support a technique if it takes exceptional effort or exceptional skill to write such programs; it merely *enables* the technique to be used." Bjarne Stroustrup [1]

Die Simulation des Systems erweist sich in der Folge als trickreich, zeigt aber die Möglichkeiten und die der objektorientierten Programmierung. So implementiert jede Komponente mit ihrer eigenen `update`-Methode ihr Verhalten. Die Aufgabe der `simulate`-Methode ist dann nur die Simulation zu steuern und z.B. dabei die Daten zu sammeln oder sie anzuzeigen.

Die Konzepte und die Möglichkeiten der objektorientierten Programmierung wurden kurz erläutert. Für Interessierte empfiehlt es sich aber eine *echte* Programmiersprache zu lernen, wie z.B. Java.

Alle M-Files zu diesem Beispiel findet man im Web:
http://www.cis.tugraz.at/ieee/ik/November-2000/objektiv.html

Schrifttum

[1] Bjarne Stroustrup: *What is Object-Oriented Programming?* (1991 revised version). Proc. 1st European Software Festival. February, 1991.