

Implementierung Akustischer Algorithmen

Iohannes m zmölnig

March 23, 2017

Pd Objekte

Funktionaler Ansatz

- Objekte manipulieren Daten: „Funktionen“
- Eingangsdaten \mapsto Ausgangsdaten

Dataflow

- Objekte/Funktionen werden **nicht** nacheinander abgerufen sondern
- Daten *fließen* durch Programm und werden modifiziert
- keine Variablen nötig

Pd Messages

- *Daten*
- **flüchtig**
 - ▶ existieren nur zum *ZeitPUNKT* des Auftretens
- asynchron / on-demand
 - ▶ User/Patch bestimmt Zeitpunkt des Auftretens

Pd Messages

MessageBox

- „eingefrorene Message“
- wird *aufgetaut* durch
 - ▶ Klicken
 - ▶ Message, die and MessageBox geschickt wird

Pd Objekte

„Objektorientierter“ Ansatz

- Objekte haben **inneren Zustand**
 - ▶ von Außen nicht sichtbar
- Methoden
 - ▶ interagieren mit innerem Zustand
- *Klasse*: Abstrakte Idee
 - ▶ z.B. „Zahlenspeicher“
- *Objekt*: Instanz einer Klasse
 - ▶ z.B. `float`

Instanzorientierter Ansatz

- jeder Code in Pd ist immer automatisch instanziiert

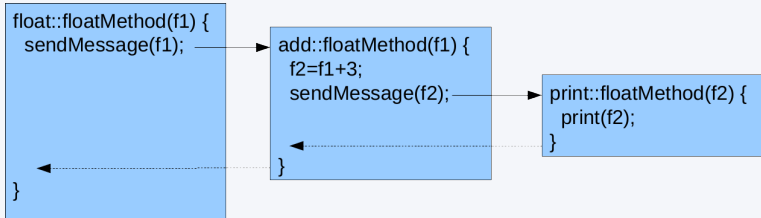
Pd Objekte: Methoden

- Messages rufen Methoden auf
- Objekt bestimmt, was mit Message (Daten) passiert
 - ▶ z.B. ob Ausgangsdaten produziert werden

```
MyObject::fooMethod(input) {  
    output=function(input);  
    sendMessage(output);  
}  
  
MyObject::barMethod(input) {  
    MyObject::state=input;  
}  
  
MyObject::guzMethod() {  
    sendMessage(MyObject::state);  
}
```

Pd Objekte: Methoden

rekursive Abarbeitung



Pd-Dispatcher „→“

- nimmt Message auf Kanal entgegen
- schickt Message an alle Objekte, die an diesem Kanal *hören*
- Messages bestimmen, welche Methode aufgerufen wird

Messages: Aufbau

- `<selector> {<atom>}`
- `<atom>`
 - ▶ Zahl (floating point)
 - ▶ Symbol (String in Hashtable)
 - ▶ Zeiger
- `<selector>`
 - ▶ Symbol
- Bsp: `connect localhost 9998`

Message: Aufbau

Theorem (Message-Selector)

jede Message hat einen Selector

- falls nicht explizit angegeben, wird ein automatischer Selector *list* (bzw. *float*) hinzugefügt

Atome

- Parser: durch Leerzeichen(`␣`) getrennt
- `float`
 - ▶ alle Zahlen in Pd
 - ▶ Single Precision (IEEE 32bit float)
 - ▶ Integers nur bis ± 16777216 . (24bit!)
 - ▶ Parser: alles was „aussieht wie eine Zahl“
- `symbol`
 - ▶ Hashtable
 - ▶ jeder „string“ wird in einer Tabelle gespeichert, und fortan nur noch indiziert
 - ▶ strings mit gleichem Inhalt werden nur einmal gespeichert
 - ▶ Tabelle wächst!
 - ▶ Parser: alles was keine Zahl ist

Messages

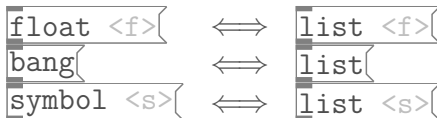
- flüchtig!
 - ▶ Messages werden vom System on-demand erzeugt und auch wieder zerstört
 - ▶ `malloc()` / `free()`

Spezielle Message-Selektoren

- bang
 - ▶ Trigger
 - ▶ immer nur <selector>, keine Daten!
 - ▶ e.g. `bang`
- float
 - ▶ immer genau ein (1!) <atom> vom Typ **number**
 - ▶ e.g. `float 3.14`
- symbol
 - ▶ immer genau ein (1!) <atom> vom Typ **symbol**
 - ▶ e.g. `symbol Algorithmen`
- list
 - ▶ allgemeine Daten
 - ▶ e.g. `list 4 Jahreszeiten`

Dispatcher

- `<selector>` einer Message wählt *Methode* eines Objektes aus
 - ▶ Objekte melden Methoden für bestimmte `<selector>`en an.
 - ▶ evtl. *catch-all* Methode
- Message an Objekt
 - ▶ wenn es Methode für `<selector>` gibt \implies aufrufen
 - ▶ sonst *catch-all*
- Implizite Konvertierung



Pd-Objectmaker

- auch *Objekte* werden als Message erzeugt
 - ▶ <selector>: Objektname
 - ▶ {<atom>}: Übergabeargument(e)
- der `objectmaker` hat eine Methode für jede *bekannte* Klasse
- *catch-all* (unbekannte Klasse)
 - ▶ versucht Klassendefinitionen von Festplatte nachzuladen