# Meilenstein I - Entwurf

Marlene Böhmer, 2547718
Andreas Meyer, 2552569

20.07.2015

## Überblick

Als Sprache haben wir uns für Java entschieden. Für die Implementierung orientieren wir uns am Fork- Join Modell. Dazu starten wir das Programm über die Klasse Supervisor. Der Supervisor enthält Objekte des Typs GraphInfo und Grid, eine von uns zu implementierende Klasse, über die erstmal nur zu sagen ist, dass sie das Gitter darstellt. Außerdem merkt sich das Programm hier die Konvergenzbedingung $\epsilon$.

Der Kern der Klasse und auch des Programms ist die Methode computeOsmose(), die Methode gliedert sich in drei wesentliche Phasen, die von einer Whileschleife nacheinander solange ausgeführt werden, bis die Konvergenzbedingungen erfüllt sind.

1. Phase: Vertikale Berechnung der Knoten und Berechnung des Outflows aller Spalten.
2. Phase: Austausch des Outflows zwischen den Spalten durch einen Iterator.
3. Phase: Berechnung der für den aktuellen globalen Iterationsschritt endgültigen Werte aller Knoten. Löschen von Spalten, die nur noch Knoten mit Wert = 0 enthalten.

In den Phasen 1 und 3 werden mehrere Threads mit start() gestartet (genau so viele wie Spalten existieren), die dann nebenläufig ihre arbeiten erledigen und terminieren. Über den Supervisor ruft der Mainthread Methoden anderer Klassen auf, die dann auf join() warten, bis alle Threads terminiert sind und fährt dann im Supervisor mit der nächten Phase(Bzw. dem nächsten globalen Iterationschritt) fort, nachdem alle Threads einen terminalen Zustand erreicht haben.

## Zu den Hilfsklassen Grid, Column, NodeEval

1. Column (extends Threads):

   Enthält als Felder

   (a) ein boolean deleteFlag= $(\sum_{i=1}^{n} Knoten) == 0$,

(b) 4 private Hashtables: oldvalues, outflowLeft, outflowRight und currentValues. Werden durch explizite Locks geschützt. Zugriff durch getter und setter.

oldValues merkt sich die Werte aller Knoten der Spalte vor dem ersten Iterationsschritt durch $oldValues = currentValues$.

(c) double $\sigma$, dass die lokale Konvergenzbedingung ($\epsilon / Gitterbreite$

(d) double valueDifference enthält nach der Berechnung von currentValues am Ende eines globalen Iterationsschrittes:

$$\sum_{i=1}^{n} (oldValues_i - currentValues_i)^2$$

Enthält Methoden:

(a) run() führt eine lokale Schleife aus, die den vertikalen Flow und den horizontalen Outflow eines globalen Iterationschrittes berechnet.

(b) computeNewValues() wird in NodeEval besprochen.

2. Grid: (implements ImageConvertible) Grid enthält ein Hashtable über alle Columns die mindestens einen Knoten mit Wert $> 0$ haben. Als Key wird der Index einer Spalte im Gitter benutzt.

   Grid enthält die Methoden

   (a) globalIteration() ruft globalIteration(Iterator iter) auf und führt rekursiv für jede Spalte column.start() aus. GlobalIteration wartet für jeden Thread mit Join() auf dessen Terminierung.

   (b) columnValueComputation() ruft columnValueComputation(Iteration iter) auf und erzeugt für jedeine e Column im Hashtable einen Thread der Klasse NodeEval, die die neuen Werte aller Knoten bzgl. des outflows evaluiert und setzt. ColumnValueComputation(Iteration iter) funktioniert ebenfalls rekursiv.

   (c) getColumn() Hilfsmethode um den horizontalen Flow eines globalen Iterationsschrittes zu realisieren.

   (d) removeColumn() entfernt alle Spalten, mit $deleteFlag = true$.

3. NodeEval (extends Thread): Ein Wrapper, um auf den Spalten Threads mit anderer Funktionalität starten zu können. Enthält als Feld eine Column, dass per Konstruktor initialisiert wird. Enthält als Methode run(), die auf Column die Methoden computeNewValues() aufruft und den Inflow mit den CurrentValues verrechnet. Hier wird die ValueDifferenz berechnet und die deleteFlag gesetzt.