Praktische Übung der Vorlesung Betriebssysteme im Sommersemester 2011 Hochschule Mannheim

Mark Albert, Henning Rohr, Patrick Beedgen, Dennis Cohen, Maiwand Haschmi

Fakultät für Informatik
Hochschule Mannheim

10.6.2011

Inhalt

- Einführung
- Gründe für die Plattformwahl
- Architektur der Anwendung
- Live Demo
- Was haben wir gelernt?

Team

- Mark Albert
- Henning Rohr
- Patrick Beedgen
- Dennis Cohen
- Maiwand Haschmi

Wo ihr uns und unser Projekt findet

https://code.google.com/p/visualscheduler/

- Dort gibt es:
- Das Standalone Java Archive
- Ein Demo Applet (ausführbar im Browser)
- und natürlich auch den Quellcode

Plattformwahl und die Vorteile

Unsere Entscheidung der Plattformwahl für das Programm ist auf Java gefallen:

- Es wird hier an der Hochschule von Anfang an gelehrt
- Es ist Plattformunabhängig
- Viele Entwicklerhilfen (Eplipse, Netbeans, verschiedene SVN Tools,... etc.)
- Möglichkeit das entwickelte Programm zu migrieren (zb. als Java Applet oder Android Application)

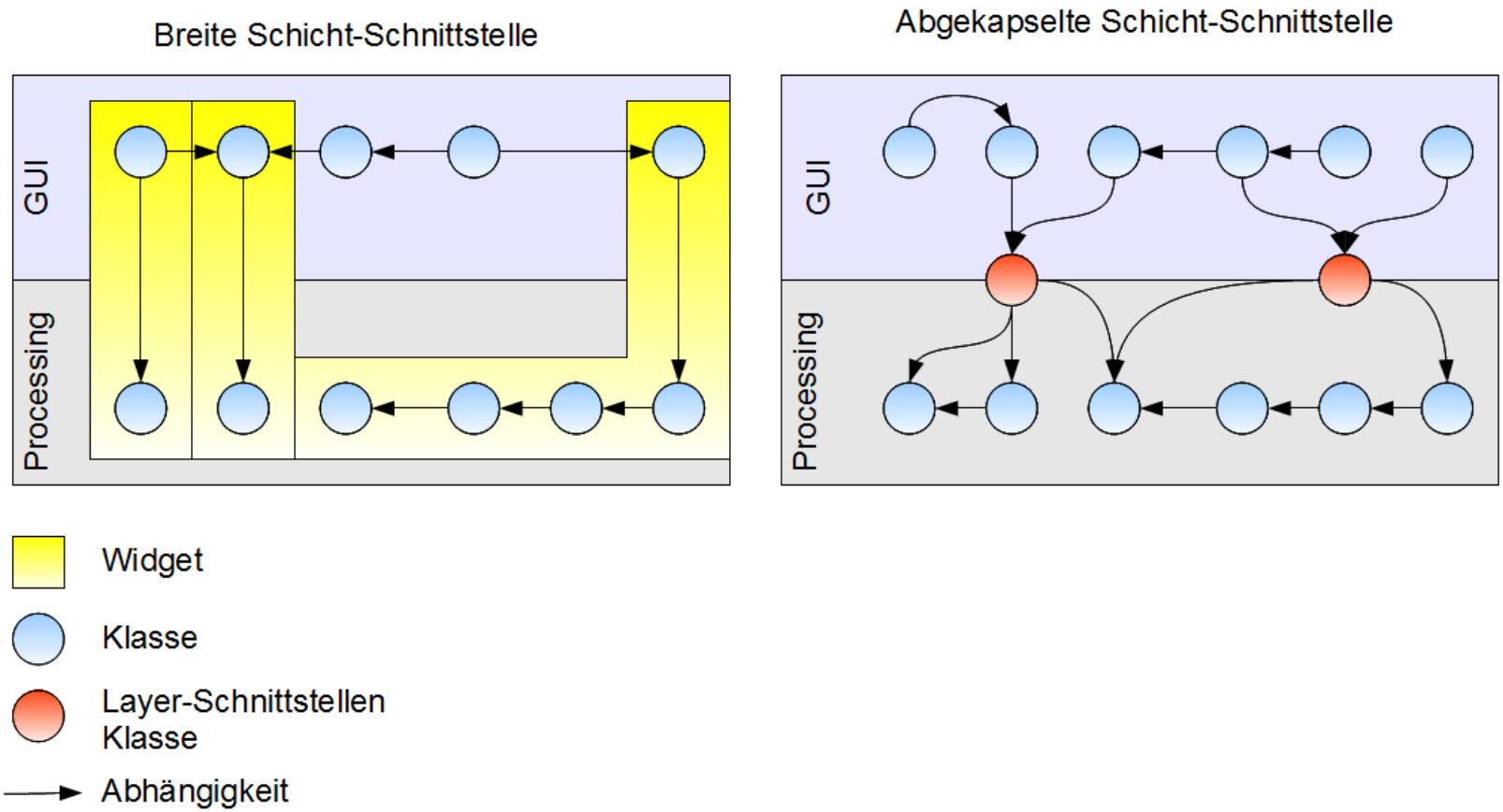
Nachteile der Plattform

- Performance schlechter als bei vergleichbaren C/C++ oä. Programmen
- Aufgrund der JVM recht hardwarefern

Architektur - Überblick

- Zweischicht Architektur mit horizontaler/breiter Layer Schnittstelle
- Einteilung in GUI und Processing
- Im weitesten Sinne eine Widget Architektur
- Ein Widget ist ein von anderen Widgets isolierbarer Codeblock bestehend aus einer GUI under einer Logik Komponenete

Architektur - Abgekapselte/Breite Architektur



Architektur - Umsetzung am Beispiel

Name	Priority	Runtime	Arrival
Process 0	1	92	41
Process 1	1	95	25
Process 2	1	6	49
Process 3	1	72	14
Process 4	1	26	9
Process 5	1	6	19
Process 6	1	12	2
Process 7	1	16	42
Process 8	1	71	23
Process 9	1	99	23

- addTableModelListener(TableModelListener) : void
- getColumnClass(int) : Class<?>
- getColumnCount() : int
- getColumnName(int) : String
- getRowCount() : int
- getValueAt(int, int) : Object
- isCellEditable(int, int) : boolean
- removeTableModelListener(TableModelListener) : void
- setValueAt(Object, int, int) : void

Abbildung: JTable/Tablemodel

- getAverageActive() : double
- getAverageActivePercentString() : String
- getAverageLatency() : double
- getAverageLatencyPercentString() : String
- getAverageRuntime() : double
- getAverageSleep() : double
- getAverageSleepPercentString() : String
- getAverageWait() : double
- getAverageWaitPercentString() : String
- getEndTime() : int
- getProcesses() : ArrayList<ProcessResult>
- update() : void

Abbildung: SchedulingResultView/Scheduling Result

Architektur - AbstractScheduler

- Basisklasse für alle Scheduler
- Veringert die Komplexität der Scheduler Implementierungen stark
- Vereinfacht das Testen indem die Gesamtzahl an Codezeilen klein gehalten wird

```
@Override
protected ScheduleResult performSchedulingBlock(ArrayList<Process> active) {
    int highestPriority = 0;
    Process best = null;
    for(Process current: active) {
        int currentPriority = current.definition.getPriority();

        if(currentPriority > highestPriority) {
            highestPriority = currentPriority;
            best = current;
        }
    }
    return new ScheduleResult(best, best.getTimeleft());
}
```

Abbildung: Priority Based Scheduling

Architektur - AbstractScheduler

- Präsentiert der Implementierung nur gestartete aktive Prozesse
- Sortiert die Prozesse nach ihrer Ankunftszeit
- Fast Teilergebnisse zusammen

```
@Override
protected ScheduleResult performSchedulingBlock(final ArrayList<Process> active) {
    return new ScheduleResult(active.get(0), active.get(0).getTimeleft());
}
```

Abbildung: First Come First Served

Architektur - Applet/App Konzept

- Applikation als Applet oder Standalone lauffähig
- Alles in einer Jar, mittels Jar-In-Jar Loader
- Keine Unterschiede im Aussehen der Funktion

```
import javax.swing.JApplet;

public class Main extends JApplet {
    public static void main(final String[] args) {
        // App entry-point
    }

    public Main() {
        // Applet entry-point
    }
}
```

Abbildung: Applet/App Konzept

Architektur - Applet/App Probleme

- In Kombination mit den Substance Look-And-Feel ist der Ansatz weitaus komplizierter
- Der Grund liegt an den Anforderungen die Substance an den Display-Container stellt
- Eine Signatur der Jar-Datei ist nötig
- Der Benutzer bekommt Warnhinweise bei einem selbstsignierten Applet
- ‘Offizielle‘ Signatur kostet Geld

Produkt-Demo

Unsere Visual Scheduler Implementierung

Lerneffekt

- Kommunikation ist sehr wichtig(Email, persönliche Absprachen...)
- Aufgabenteilung teilweise etwas schief gelaufen(Aufgrund schlechter Kommunikation... s.o!)
- gutes Design erleichtert ungemein den Aufwand des Programmierens
- besseres Verständnis der Funktionsweise von Schedulingverfahren

Fragen?