Technische Universität München
Informatik VIII
Netzarchitekturen und Netzdienste
Prof. Dr.-Ing. Georg Carle

Alexander Klein, Nils Kammenhuber

Übungen zur Vorlesung Diskrete Simulation

Übungsblatt 2, SS 2010

Abgabe: 22. Juni 2010 (bis 10 Uhr am Lehrstuhl I-8, Finger 03.05)
Die Aufgaben können jeweils von zwei Studenten gemeinsam bearbeitet werden.

Aufgabe 1 — Programmieren eines einfachen Event-Simulators (100 Punkte)

Ziele: *Vertiefung des einfacher Simulations-Konzepte, systematische Herangehensweise bei objektorientierter Softwareentwicklung, Stöbern in der Schatzkammer namens java.util.*

Sowohl ein rein eventbasierter Simulator (im Gegensatz zu einem prozessorientierten Simulator) als auch unser einfaches Warteschlangenmodell ($GI/GI/1/\infty$) sind verhältnismäßig einfache Konzepte. In dieser Aufgabe sollen sie miteinander kombiniert werden: Ziel der Aufgabe ist die Implementierung eines einfachen diskreten Event-Simulators, welcher ein solches Warteschlangenmodell simulieren kann. Der Simulator sollte in einer objektorientierten Sprache geschrieben werden; wir empfehlen hierfür Java.

Bitte lassen Sie sich vom Textumfang dieser Aufgabe nicht abschrecken: wir geben absichtlich relativ viel vor, um Sie vor möglichst vielen potenziellen konzeptuellen Holzwegen oder umständlichen Herangehensweisen zu bewahren.

a) **Implementierung Event-Kernel** [25]
Entwerfen Sie zunächst das Grundgerüst eines abstrakten diskreten Event-Simulators (*Simulator-Kernel* oder *Event-Engine*), so wie er in der Vorlesung vorgestellt wurde (2. Vorlesung sowie Erklärung der Aufgabenstellung dieses Übungsblattes). Dieses Grundgerüst sollte mindestens über folgende Klassen verfügen:

- `Event` – abstrakte Klasse, die ein Ereignis simuliert. Hat eine Event-Zeit und kann ausgeführt werden. Beachten Sie bei der Wahl des Datentyps für die Simulationszeit, wie die Zufallszahlen erzeugt werden (nächste Teilaufgabe).
- `SimulationTerminationEvent` – eine von `Event` abgeleitete Klasse, welche bei ihrer Ausführung die Simulation sofort beendet und die Möglichkeit bietet, ggf. Statistiken o. ä. auszugeben.
- `EventChain` – dient zum Aufnehmen zukünftiger Events; bietet eine Operation zum Anschauen und Entfernen des aktuellsten Events.
  Tip: Im Paket `java.util` gibt es für sehr viele Anwendungsfälle eine geeignete Datenstruktur – für unseren Fall mindestens eine, seit JDK 1.6 sogar mindestens zwei.
- Eine Hauptklasse `Simulator`, welche von der Kommandozeile liest, für wieviele Sekunden die Simulation laufen soll, einen entsprechenden `SimulationTerminationEvent` plant und dann die Hauptroutine der Simulation ausführt (vgl. Vorlesung).

b) **Implementierung Warteschlangenmodell** [30]
Erweitern Sie nun den abstrakten Simulator so, dass er unser Warteschlangenmodell simulieren kann. Erstellen Sie dazu folgende Klassen:

- `CustomerArrival` – Unterklasse von `Event`. Simuliert die Ankunft eines Kunden und erstellt einen entsprechenden Nachfolger-Event in der Zukunft (Stichwort Zwischenankunftszeit / interarrival time). Verwenden Sie für die Zwischenankunftszeiten zunächst einmal `Math.random()*11.0`.
- `ServiceCompletion` – Unterklasse von `Event`. Simuliert, dass die Bedieneinheit soeben ihren aktuellen Job fertiggestellt hat (Bedienzeit / service time) und nun bereit ist für den nächsten Kunden. Verwenden Sie für die Bedienzeiten zunächst einmal `Math.random()*10.0`.
- ggf. weitere Klassen, um den Zustand von Warteschlange und/oder Bedieneinheit zu repräsentieren. Behalten Sie bei deren Entwurf auch die nächste Teilaufgabe im Auge.

Implementierungsfrage: Wie implementiert man am besten den „Event", dass ein Kunde aus der Warteschlange herausgenommen wird und von der Bedieneinheit bedient wird? Man benötigt dafür keine eigene Event-Klasse; sie würde den Code sogar eher verkomplizieren.

c) **Implementierung Statistik-Ausgaben:** [25]
Unser Simulator ist bislang völlig witzlos, weil er zwar simulieren kann, aber keinerlei Ausgaben produziert. Erweitern Sie den Simulator nun so, dass er folgende Statistiken berechnet und am Ende der Simulation ausgibt:

- Durchschnittliche Wartezeit eines Kunden in der Warteschlange
- Durchschnittliche Durchlaufzeit für Kunden (= die Zeit, die er in der Warteschlange verbringt plus seine Bedienzeit)
- Maximaler Füllstand der Warteschlange (maximale Anzahl Kunden, die gleichzeitig warten)
- Durchschnittliche Auslastung der Bedieneinheit (0: hat nie was zu tun; 1: ist permanent beschäftigt)

d) **Auswertung:** [15]
Lassen Sie die Simulation fünf Mal laufen (Simulationszeit jeweils mindestens 10 000 Sekunden). Vergleichen und interpretieren Sie die Ergebnisse.

e) **Kleiner Denkanstoß:** [5]
Unser obiger Implementierungsvorschlag für die Generierung von Zufallszahlen ist gleich in mehrfacher Hinsicht schlecht. Inwiefern?