Modell eines Insel-Callshops

2. Projekt zu Modellierung und Simulation

Daniel Graf, Dimitrie Diez, Arne Schöntag, Peter Müller

Inhaltsverzeichnis

1 Einführung **1**

2 Beschreibung des Modells **2**

3 Anforderungen/Requirements **2**

4 Softwaredesign **2**
4.1 Package time 3
4.2 Package queue 5
4.3 Package distribution 6
4.4 Package domain 7
4.5 Package calculation, log 8
4.6 Package config, modellbildung 9

5 Implementierung **10**

6 Softwaretest **10**

7 Simulation **10**

8 Fazit **10**

1 Einführung

Simulationen haben in der moderne einen sehr hohen Stellenwert erlangt, da durch sie zahlreiche, oftmals sehr genaue, Zukunftsprognosen erstellt werden können. Das Thema dieser Studienarbeit ist die Simulation eines Callshops, bzw. des Telefons in einem Callshop, in einem Inseldorf. Dieser wird für günstige Telefonate ins Ausland verwendet.

Mit Hilfe der Simulation sollen anschließend Aussagen bezüglich der zukünftigen Auslastung des Telefons getroffen werden können. Konkret wird die Fragestellung betrachtet, ob ein zweites Telefon im Callshop sinnvoll oder unnötig ist.

Als Basis für die erläuterten Prognosen werden folgende Werte der Simulation ermittelt:

- mittlere Länge der Warteschlage (anstehende Kunden)
- mittlere Wartezeit bis zum Telefonat
- mittlere Verweildauer im Callshop (Wartezeit + Gesprächsdauer)
- Mittlere Auslastung des Telefons

2 Beschreibung des Modells

Für die Simulation des Insel-Callshops wird ein Warteschlagenmodell mit jeweils einem Client und Server verwendet. Jede Person die den CallShop betritt wird durch einen neuen Client repräsentiert. Das Telefon des Shops ist durch den Server dargestellt. Möchte eine Person das Telefon zu einem Zeitpunkt benützen, zu dem bereits eine andere Person telefoniert, muss sie sich hinten anstellen und warten, bis die Person ihr Telefonat beendet hat. Im Modell wird dieses durch eine Warteschlage (Queue) realisiert, in die sich die wartenden Clients einordnen und nach dem FIFO (first in first out) Prinzip bedient werden.

Im Modell werden sowohl die Ankunftszeiten der Clients, als auch die Dauer der Telefonate durch eine negative Exponentialverteilung beschrieben, da diese sehr nah an den real beoachteten Verhalten liegt. Der mathematische Hintergrund liegt in der Eigenschaft der Exponentialfunktion zugrunde. Die Exponentialverteilung ist die einzige kontinuierliche Verteilung, welche zugleich die Markoveigenschaft, die sogenannte Gedächtnislosigkeit, erfüllt. Diese besagt, dass die seit dem letzten Ereignis vergangene Zeit (in diesem Beispiel Anrufer) keinen Einfluss auf die Verteilung der Zeit bis zum nächsten Ereignis (bis zum nächsten Anruf) hat. Quelle: http://www.mathepedia.de/Exponentialverteilung.aspx

3 Anforderungen/Requirements

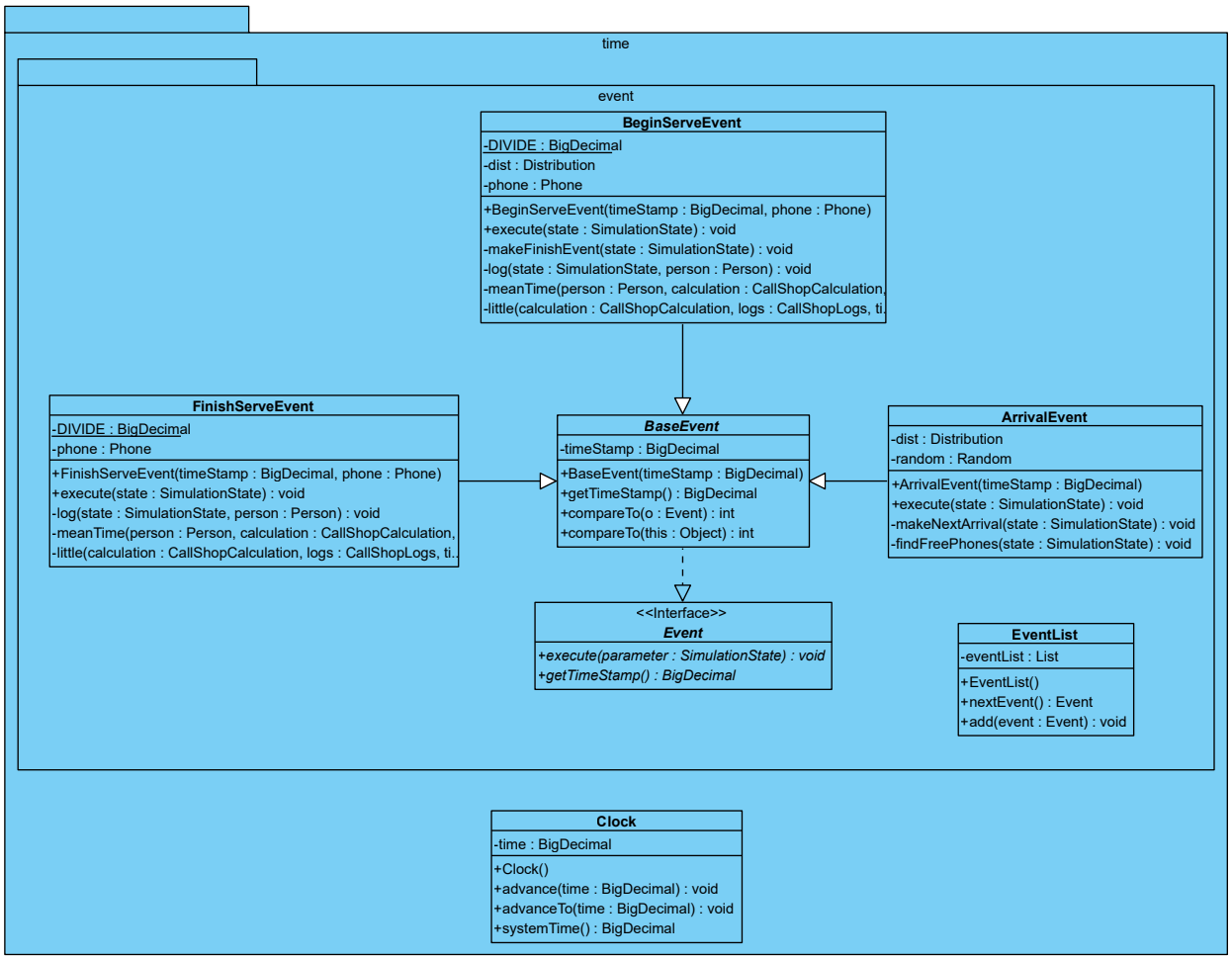
4 Softwaredesign

Der Aufbau der Anwendung wurde im Team diskutiert und anschließend mittels UML spezifiziert.

Grundsätzlich wurde eine Ereignisorientierte Simulation (Discrete Event Simulation) mit einer Eventliste gewählt. Ursprünglich war geplant die Berechnung mit Mathematica auszuführen. Dies stellte sich aber als nicht performant und komplizierter als notwendig

heraus. Alle geforderten Berechnungen wurden dann ebenfalls in das Java Programm eingeplant.

4.1 Package time



In diesem Unterpaket finden sich alle notwendigen Klassen für die Ereignisorientierte Simulation. Die `Clock` hält die Simulationszeit und kann nur vorgestellt werden.

Das Interface `Event` definiert ein Ereignis in der Simulation. Jedes Ereignis hat einen Zeitpunkt an dem es auftritt, sowie Logik, die zu diesem Zeitpunkt ausgeführt werden soll und potentiell den ganzen Zustand der Simulation verändern kann. Die `EventList` kann Ereignisse aufnehmen und immer das nächste Event mit den frühesten Zeitpunkt liefern.

Die konkreten Events `Arrival, BeginServe, FinishServe` beschreiben die Ereignisse in der Callshop Problemstellung.

Abbildung 1 zeigt den Verlauf des Ereignisses für Arrival. Zunächst werden freie Telefone gesucht. Wird ein freies Telefon gefunden wird ein BeginServeEvent mit aktueller Zeit erstellt. Eine neue Person wird erstellt, ihre Ankunftszeit gespeichert, und die Person wird in die Schlange eingefügt. Zum Schluss wird das nächste ArrivalEvent erstellt, um fortlaufend neue Ankünfte zu haben.

Der Verlauf von BeginServe ist in Abbildung 2 zu sehen. Die nächste (VIP oder normale) Person wird je nachdem, ob es sich um ein VIP Phone oder nicht handelt, aus der Liste geholt. Die Zeit vom Bearbeitungsbeginn wird für die Person gespeichert. Das FinishEvent wird direkt ermittelt und in die EventList eingetragen.

Das Ereignis FinishServe ist in Abbildung 3 dargestellt. Die Person wird aus dem Phone entfernt. Ihre Endzeit wird gespeichert. Ist die Warteschlange nicht leer, wird direkt ein neues BeginServeEvent erstellt.

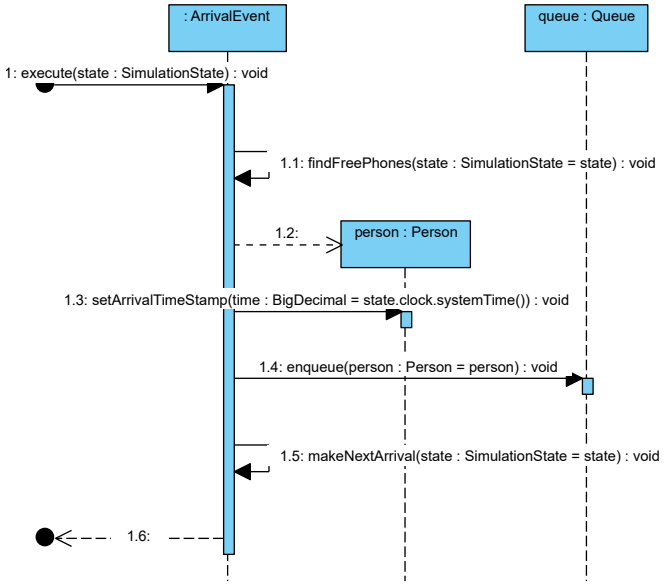


Abbildung 1: Sequenzdiagramm für die Ausführung eines ArrivalEvent

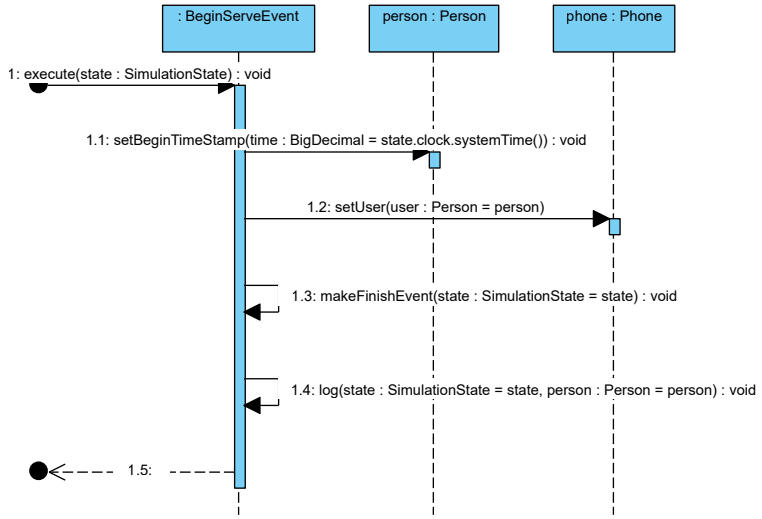
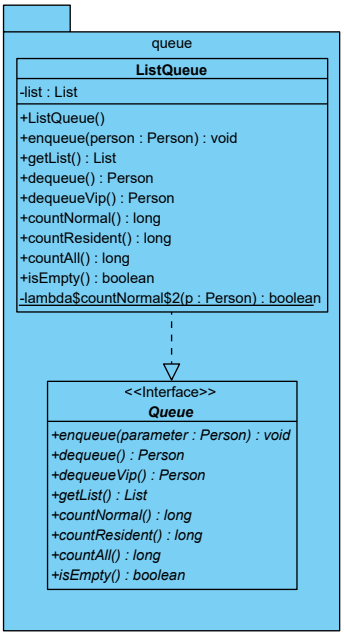


Abbildung 2: Sequenzdiagramm für die Ausführung eines BeginServeEvent

4.2 Package queue



Die `Queue` ist eine FIFO-Warteschlange mit zugehörigen Operationen. Optional kann direkt auch der erste Resident (VIP) aus der Schlange entfernt werden.

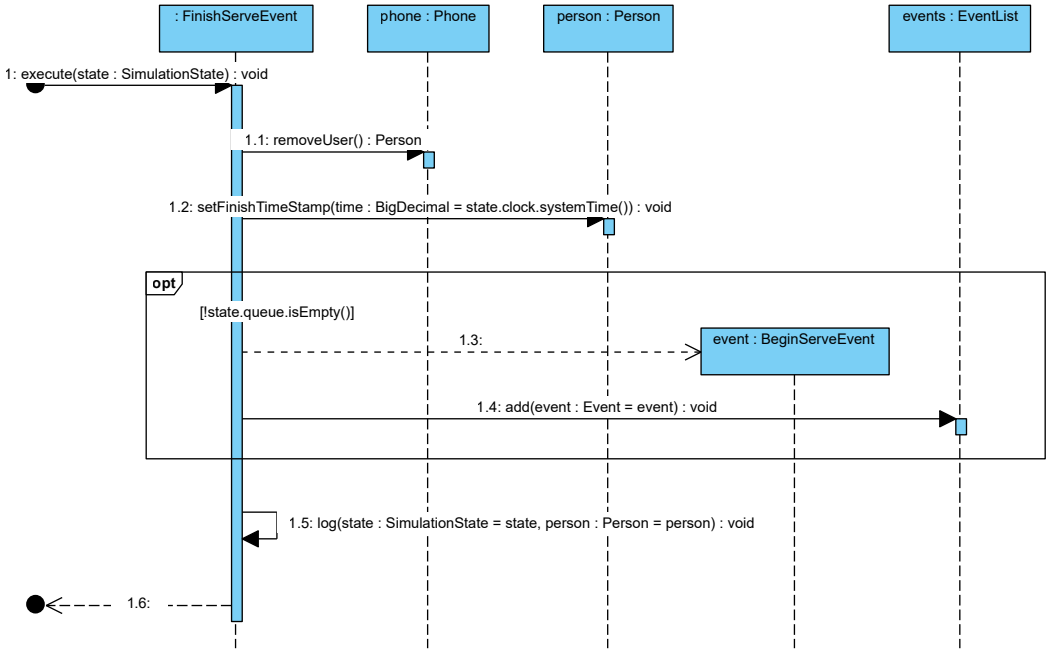
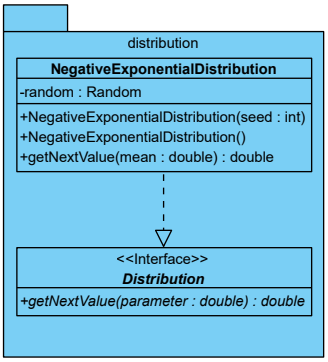


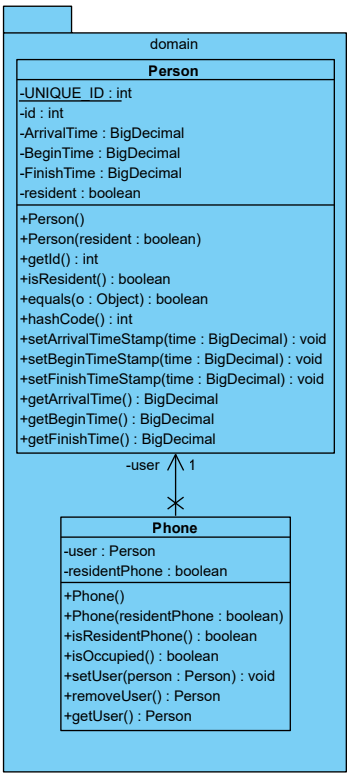
Abbildung 3: Sequenzdiagramm für die Ausführung eines FinishServeEvent

4.3 Package distribution



Die Verteilung ist grundsätzlich eine negative Exponentialverteilung. Die Zufallszahlen können über einen Seed erstellt werden.

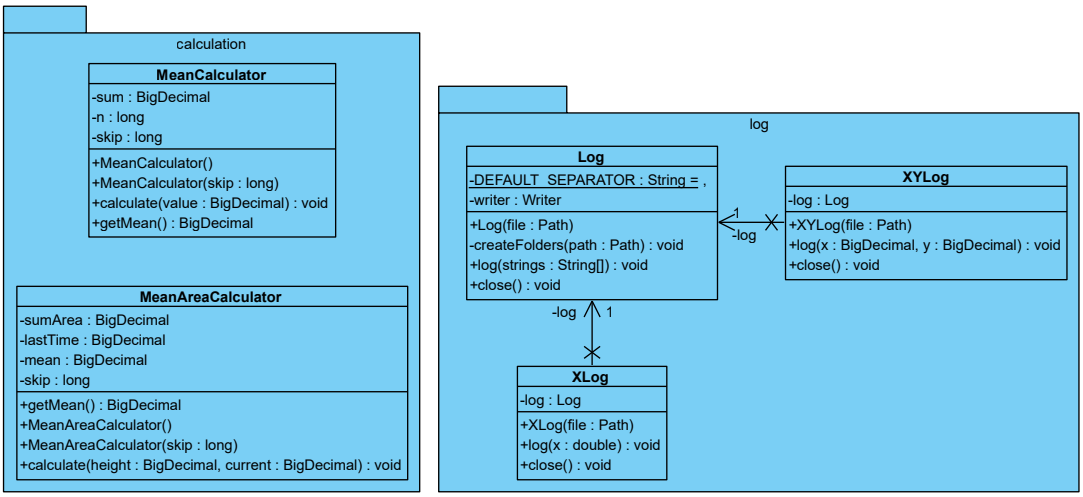
4.4 Package domain



`domain` beschreibt alle notwendigen Datenstrukturen aus der Domäne des Callshops. Hier gibt es Personen und Telefone. Jede neue Person hat eine einzigartige ID und im Laufe der Simulation merkt sie sich ihre persönlichen Zeitpunkte.

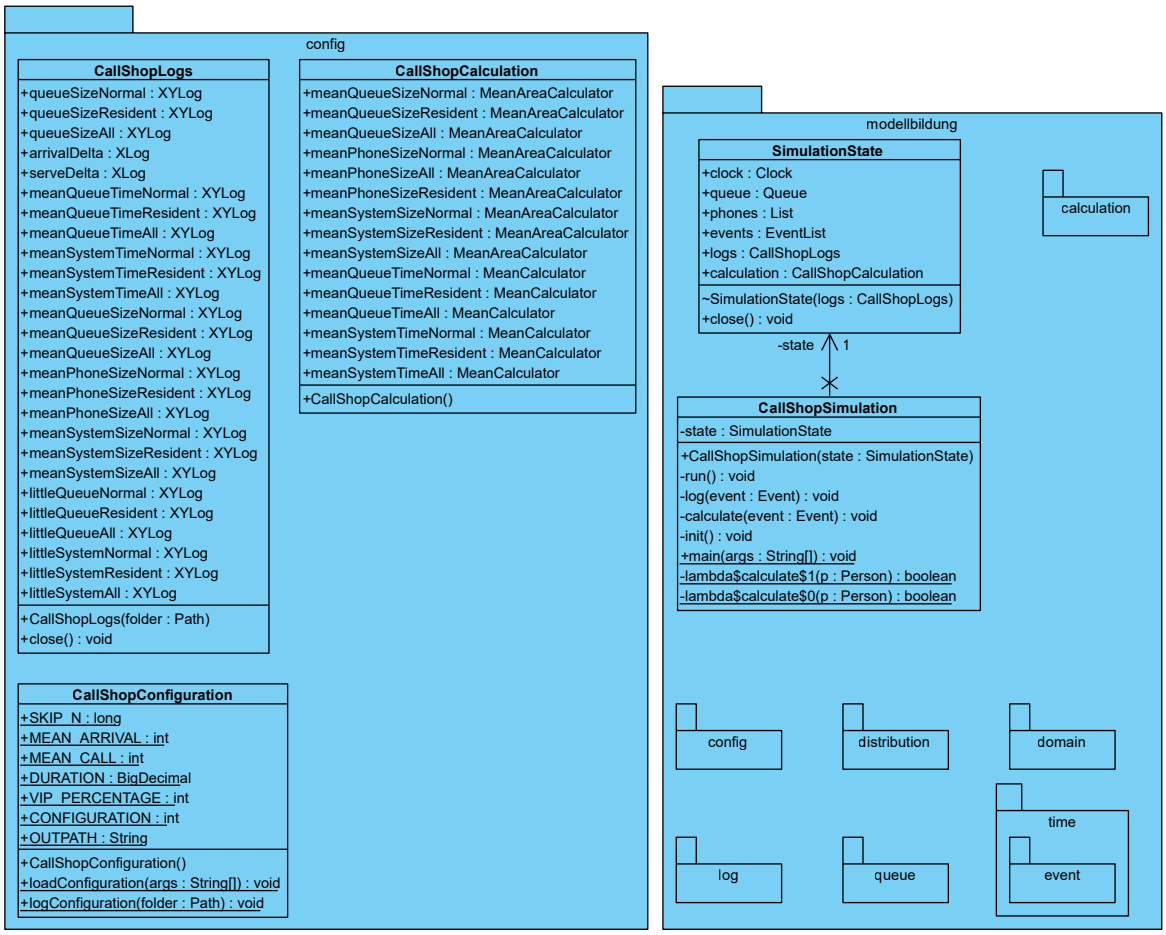
Eine Person kann optional ein Resident (Vip) sein. Eine Telefon kann optional nur Residents (VIPs) annehmen.

4.5 Package calculation, log



Die allgemeine Berechnung für Mittelwerte ist in `calculation` gelöst. Mit `log` können Ergebnisse in ein CSV Dokument gespeichert werden.

4.6 Package config, modellbildung



In `config` ist die Konfiguration der Simulation zu finden. Die Logs und Mittelwert-Rechner für die aktuelle konkrete Simulation sind ebenfalls hier definiert.

Das Paket `modellbildung` beinhaltet alle zuvor beschriebenen Pakete und enthält die Simulationsklasse zum Starten sowie den groben Ablauf der Simulation. Der `SimulationState` hält den aktuellen Zustand der Simulation.

5 Implementierung

6 Softwaretest

7 Simulation

8 Fazit