technische universität dortmund

Fakultät für Informatik
Lehrstuhl 4

Falko Bause, Iryna Dohndorf

Sommersemester 2017

Modellgestützte Analyse und Optimierung
Übungsblatt 7

Ausgabe: 29.05.2017, **Abgabe:** 05.06.2017

Aufgabe 7.1: Berechnung von Konfidenzintervallen (4 Punkte)

Nachfolgend finden Sie die Beobachtungen einer Leistungsgröße Y (zu einem festen Zeitpunkt), die aus 30 replizierten Simulationsläufen mit unterschiedlichen Zufallszahlen stammen.

4.35	5.29	4.43	5.54	6.08	5.34
6.84	4.87	5.63	4.80	5.00	6.40
1.95	3.40	4.77	4.79	4.29	5.85
5.13	4.21	3.55	6.56	5.60	3.61
6.64	3.95	4.57	6.02	2.98	3.62

a) Argumentieren Sie umgangssprachlich warum die 30 Beobachtungen von Y als Realisierungen *unabhängiger* Zufallsvariablen angesehen werden können.

Bestimmen sie ein 90% sowie ein 95% Konfidenzintervall für den Erwartungswert $E[Y]$ mittels

b) der Tschebyscheff'schen Ungleichung

c) der Approximation über die Normalverteilung (siehe Tabelle mit $\nu = \infty$)

d) der t-Verteilung (siehe Tabelle).

ν	0.8	0.9	0.95	0.975	0.99
28	0.855	1.313	1.701	2.048	2.467
29	0.854	1.311	1.699	2.045	2.462
30	0.854	1.310	1.697	2.042	2.457
∞	0.842	1.282	1.645	1.960	2.326

Kritische Werte der t-Verteilung mit ν Freiheitsgraden

Aufgabe 7.2: Batch Means (4 Punkte)

Eine Simulation liefere für den zu messenden Wert die folgenden 32 Werte:

15.9	15.1	5.3	4.4	1.7	22.1
2.0	8.2	0.1	1.5	23.9	10.8
21.3	21.7	14.2	5.4	4.4	18.6
5.0	1.8	8.8	1.0	11.3	4.7
15.3	23.7	12.9	14.5	40.3	4.8
5.6	9.4				

Es sei angenommen, dass die Werte identisch verteilt, aber nicht unabhängig voneinander sind. Wenden Sie das Batch Means Verfahren zur Bestimmung eines 98% Konfidenzintervalls an. Verwenden Sie die Batchgröße 4!

Aufgabe 7.3: Agentenbasierte Simulation mit AnyLogic (4 Punkte)

In dieser Aufgabe soll ein Service-Center, das mit mehreren Fahrzeugen Wartungen und Reparaturen an Maschinen in Fabriken vornimmt, modelliert werden. Die Modellierung soll agentenbasiert erfolgen.

Die wichtigsten Funktionen zur agentenbasierten Modellierung befinden sich in der Agenten-Palette. Sobald Sie ein *Agent*-Objekt in das Modellfenster ziehen, öffnet sich ein Wizard, mit dem eine genauere Spezifikation vorgenommen werden kann. AnyLogic bietet hier drei Möglichkeiten: Eine Population von Agenten, einen einzelnen Agenten oder einen Agenten-Typ. In den ersten beiden Fällen werden direkt ein oder mehrere Agenten erzeugt; der Agenten-Typ kann genutzt werden, um erst einmal nur abstrakt das Verhalten eines Agenten zu beschreiben, ohne dass direkt Realisierungenn dieses Agenten erzeugt werden. Der Typ entspricht also programmiertechnisch einer Klassenbeschreibung, während die Population den erzeugten Objekten dieser Klasse entspricht.

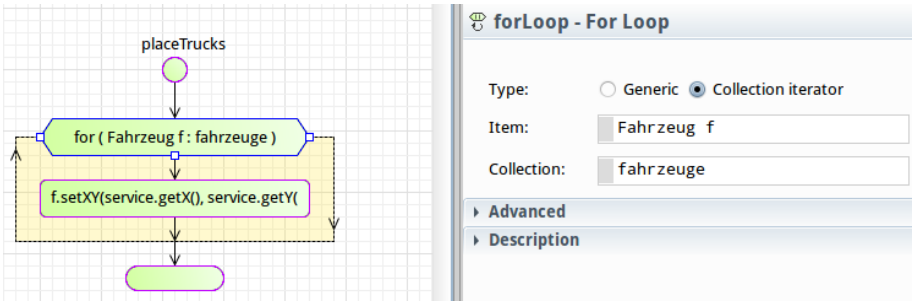
Unser Modell wird von allen drei gebotenen Möglichkeiten Gebrauch machen. Das Modell soll aus einem Service-Center bestehen, in dem fünf Fahrzeuge bereit stehen, um Wartungs- und Reparatur-Arbeiten in 15 Fabriken vorzunehmen. Wenn eine Wartung ansteht, benachrichtigen die Fabriken das Service-Center, das ein Fahrzeug losschickt. Sollte die Wartung nicht rechtzeitig ausgeführt werden, wird die Maschine in der Fabrik beschädigt und eine aufwendigere Reparatur ist nötig.

1) Anlegen der Agenten:
   - Erzeugen Sie einen Agenten für das Service-Center (Typ-Name *Service*, Agenten-Name *service*), und zwei Populationen für die Fabriken (Typ-Name *Fabrik*, Population-Name *fabriken*) und die Fahrzeuge (Typ-Name *Fahrzeug*, Population-Name *fahrzeuge*).
   - Wählen Sie passende Grafiken für die Animation, eine passende Größe für die Umgebung und *Apply random layout*, um die Agenten zufällig im Simulationsbereich anzuordnen.
   - Wir werden später sehen, dass wir für Nachrichten, mit denen Fabriken eine Wartung anfordern, ebenfalls ein Objekt benötigen, das durch einen Agenten realisiert wird. Erzeugen Sie hierfür nur einen Agenten-Typen mit dem Namen *ServiceReq*.

AnyLogic bietet umfangreiche Möglichkeiten das Verhalten von Agenten zu modellieren. *State charts* sind eine grafische Möglichkeit, um Verhaltensmuster der Agenten darzustellen. Diese bestehen aus Zuständen, in denen sich ein Agent befinden kann, und Transitionen, die Übergänge zwischen den Zuständen beschreiben. Transitionen können durch eingehende Nachrichten, den Ablauf eines Zeitintervalls oder die Ankunft eines Agenten an einem bestimmten Ort ausgelöst werden. Für das Verhalten des Agenten in einem Zustand oder bei einer Transition können Aktionen ausgeführt werden, die mit Hilfe von Source Code definiert werden. Zusätzlich können Agenten über Parameter, Variablen und *Collections* (Arrays und Listen) von Variablen verfügen. Neben den Aktionen lässt sich Source Code auch mit Hilfe von Funktionen und *Action charts* definieren. *Action charts* stellen eine grafische Möglichkeit zur Eingabe von Code dar.

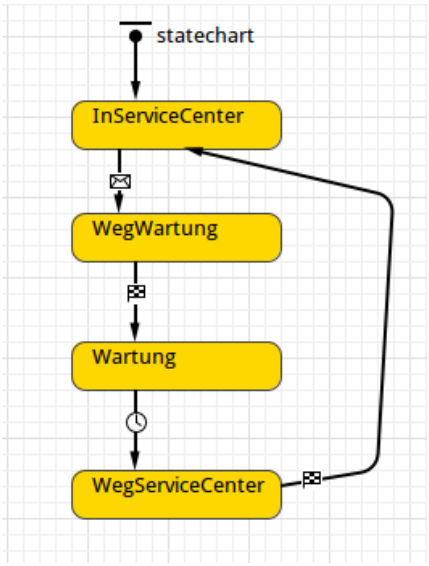
2) Verhalten der Agenten:

- Die Agenten werden zufällig im Simulationsbereich angeordnet. Für die Fabriken und das Service-Center ist dies erwünscht, die Fahrzeuge sollen sich allerdings zu Beginn im Service-Center befinden. Dies kann mit Hilfe eines *Action charts* im *Main*-Agenten realisiert werden:



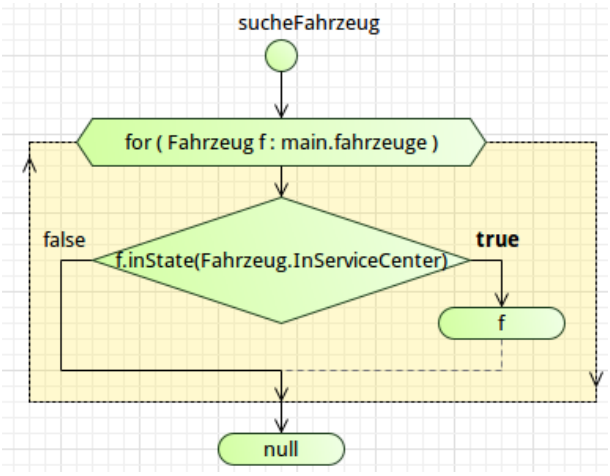
  Die Abbildung zeigt das Chart, das aus einer For-Schleife und einem Code-Block besteht, und die Eigenschaften der Schleife. Damit der Code ausgeführt wird, geben Sie in den Eigenschaften des Agenten als Aktion unter *On startup placeTrucks();* ein.

- Das Verhalten der Fahrzeuge wird mit einem *Statechart* modelliert:

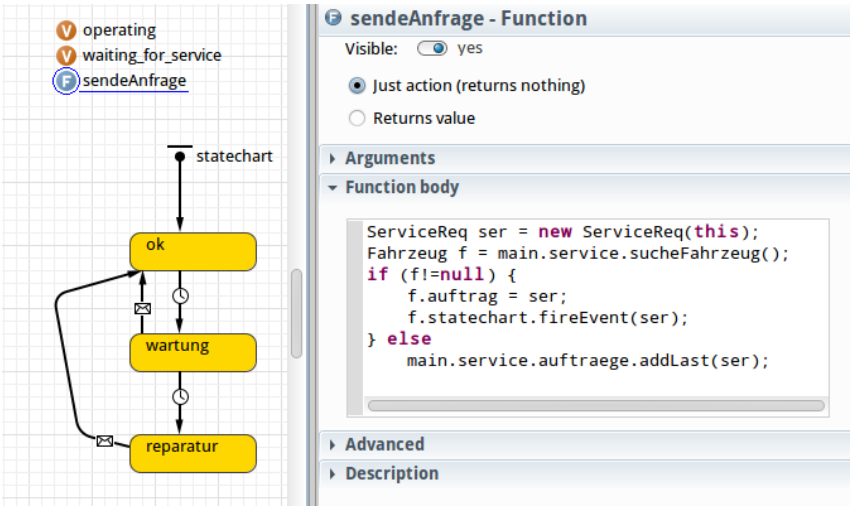


  Ein Fahrzeug befindet sich zu Beginn im Service-Center. Durch den Eingang einer Nachricht (Wartungsauftrag) macht es sich auf den Weg. Nach der Ankunft in der entsprechenden Fabrik beginnt die Wartung. Nachdem die für die Wartung benötigte Zeit verstrichen ist, macht es sich auf den Rückweg zum Service-Center. Das grundsätzliche Verhalten der Fahrzeuge ist damit beschrieben. Aktionen, die in einem Zustand oder durch eine Transition ausgeführt werden sollen, ergänzen wir später. Legen Sie zusätzlich noch eine Variable mit dem Namen *auftrag* vom Typ *ServiceReq* an. Hiermit kann ein Fahrzeug seinen aktuellen Auftrag speichern.

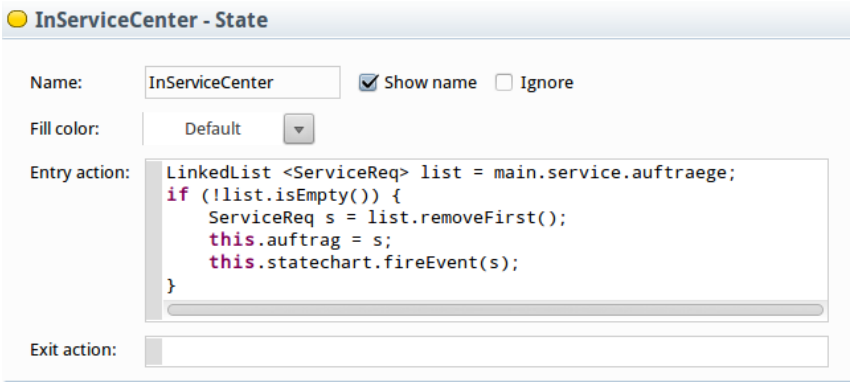
- Das Service-Center muss die eingehenden Wartungsaufträge verwalten und auf die Fahrzeuge verteilen. Zur Speicherung der eingehenden Aufträge legen Sie eine *Collection* mit dem Namen *auftraege* an. Die *LinkedList* speichert Elemente vom Typ *ServiceReq*. Für die Verteilung auf die Fahrzeuge können wir erneut ein *Actionchart* nutzen:



- Der Agenten-Typ *ServiceReq* erhält einen Parameter vom Typ *Fabrik* mit dem Namen *fabrik*.
- Die Fabrik erhält zwei boolsche Variablen, die angeben, ob die Maschinen funktionstüchtig sind und ob eine Service-Anfrage gestellt wurde. Zusätzlich erhält sie eine Funktion, mit der Service-Anfragen an den Service-Center gestellt werden. Das weitere Verhalten modellieren wir mit einem *Statechart*. Die folgende Abbildung zeigt den *Statechart* und die Funktion *sendeAnfrage* zum Anfordern eines Wartungsauftrags:



- Ergänzen Sie die *Statecharts* der Agenten *Fahrzeug* und *Fabrik* mit sinnvollen Aktionen für die Zustände und Transitionen. Die folgende Abbildung zeigt die *Entry Action*, die ausgeführt wird, wenn ein Fahrzeug zum Service-Center zurückkehrt, um einen neuen Auftrag aus der Liste des Service-Centers zu übernehmen:



3) Verfeinerung des Modells:
    - Was fehlt noch für einen vernünftigen Ablauf? Ergänzen Sie das Modell mit sinnvollen Angaben für die Geschwindigkeiten der Agenten, die Skalierung des Simulationsbereichs und Zeiten für Wartungen und Ausfälle.