PS Einführung Simulation

Simulationszeit (Darstellung, Funktionalität, Scheduling von Ereignissen/Prozessen)

Dario Hornsteiner, Sofia Bonini, Sabine Hasenleithner

Universitaet Salzburg, 5. Mai 2021

1 Allgemeine Erklärung:

Die Simulationszeit ist eine fiktive Modellzeit, unabhängig von der realen Zeit und der Ausführungsdauer einer Simulation. Bei der Diskreten Ereignis-Simulation springt die Simulationszeit von Ereignis zu Ereignis.Ein Ereignis löst maßgebliche relevante Zustandsänderungen bei einer oder mehreren Entitäten aus. Dabei ist ein Eintritts- oder Ereigniszeitpunkt ein Zeitpunkt in der Simulationszeit.
Es gibt eine orthogonale Beziehung zwischen Simulationszeit und Rechenzeit. Die Simulationszeit schreitet sprunghaft voran, da sie Ereignisse abarbeitet. Sie ist monoton steigend. Rechenzeit wird zur Ausführung am Rechner benötigt, wobei Ereignisse Rechenzeit verbrauchen, aber keine Simulationszeit. Intervalle zwischen Ereignissen verbrauchen Simulationszeit, aber keine Rechenzeit.
In der Softwareumsetzung wird eine Variable verwendet, die die aktuelle Simulationszeit darstellt.
Der Scheduler ist die Steuerung der Ablaufkontrolle, eine sequenzielle Abarbeitung der nach Ereigniszeitpunkten geordneten Ereignisse. Die Simulationszeit springt von Ereigniszeitpunkt zu Ereigniszeitpunkt, nur dabei wird die Simulationsuhr weiter gesetzt.

Es gibt mehrere wichtige Klassen für die Simulationszeit, folgende befinden sich im package *desmoj.core.simulator*

2 Desmoj Klassen betreffend Simulationszeit

2.1 TimeSpan

.
Hierbei handelt es sich um Zeitspannen der Simulationszeit. Jede Zeitspanne der Simulationszeit wird von einem individuellen Objekt dieser Klasse repräsentiert und bietet eigene Methoden für arithmetische Operationen und Vergleiche. Diese Klasse versichert, dass nur gültige Zeitspannen der Simulationszeit erstellt werden.
Es gibt mehrere Konstruktoren, und zwar:
Constructor =

- *TimeSpan(double duration)*
- *TimeSpan(double duration, java.util.concurrent.TimeUnit unit)*
- *TimeSpan(long duration)*

– *TimeSpan(long duration, java.util.concurrent.TimeUnit unit)*

Man kann also im Grunde aussuchen, ob man einen Long oder Double Wert verwendet, und ob man eine bestimmte Einheit mitgeben möchte. Wenn man keine Einheit angibt, wird die Einheit der Referenzzeit verwendet. Als duration übergibt man die Dauer der Zeitspanne.

2.2 TimeInstant

java.lang.Object
└ *desmoj.core.simulator.TimeInstant*

.

Ein TimeInstant ist ein Zeitpunkt der Simulationszeit. Dieser wird verwendet, um einen Zeitpunkt der Simulationszeit zu zeigen, an dem sich der Zustand des Modells ändert. Jeder Zeitpunkt der Simulationszeit wird mit einem eigenen Objekt dieser Klasse repräsentiert und bietet auch seine eigenen Methoden für arithmetische Operationen.

Es gibt mehrere Konstruktoren, bei denen man unter anderem einen Double Wert als Zeit übergeben kann oder ein *"Date"* Objekt, ein *"Calender"* Objekt oder eine Referenz. Der für uns relevanteste ist folgender:
Constructor = *TimeInstant(long time, java.util.concurrent.TimeUnit unit)*
Der Konstruktor konstruiert ein Objekt mit dem gegebenen Wert in der gegebenen Zeiteinheit.

2.3 SimClock

java.lang.Object
└ *java.util.Observable*
  └ *desmoj.core.simulator.SimClock*

.

Die Simulationsuhr ist, grob gesagt, die Kapselung der Simulationszeit. Sie zeigt die tatsächliche Simulationszeit. Die Simulationszeit kann von jedem Objekt abgefragt werden, aber kann nur von dem Scheduler gesetzt werden, der für das Modell zuständig ist.
Die Simulationsuhr erweitert die java.util.Observable Klasse, dies lässt zu, dass Observer sich bei der Simulationuhr registrieren können und somit immer informiert werden, wenn sich die Simulationszeit ändert. Dies kann dazu verwendet werden, einen komplett automatischen Statistikzähler zur Verfügung zu stellen. Jedes mal, wenn die Simulationszeit sich ändert, wird ein Zähler, der bei der Simulationsuhr registriert ist, benachrichtigt und kann nun den Wert abfragen, den er überwacht. Auf diese Weise benötigt man keinen expliziten Aufruf, dass der Zähler seinen Wert aktualisieren muss. Dies könnte jedoch die Performance schwächen, im Vergleich zu expliziten Aufrufen den Wert zu aktualisieren, da der Wert, der beobachtet wird, sich nicht immer ändert, wenn sich die Simulationszeit ändert.

Constructor = *SimClock(java.lang.String name)*
Dieser Konstruktor konstruiert eine Simulationsuhr mit keinen festgelegten Parametern. Als Default wird die Simulationszeit auf 0 gesetzt.

2.4 Scheduler

Prinzipiell soll das Scheduling die Simulation steuern. Jedoch ist die Funktion je nach Modellierungsstil unterschiedlich. Wir haben in der Vorlesung bereits zwei Modellierungsstile kennengelernt. Den ereignisorientierten und den prozessorientierten.

- Ereignisorientierter Modellierungsstil: Das Scheduling steuert hier den Ablauf der Simulation. Die Ereignisse werden nach Ereigniszeitpunkten geordnet und durch die Simulation abgearbeitet, wobei nur diese Zeitpunkte simuliert werden, zu welchen auch wirklich ein Ereignis stattfindet (also nicht die gesamte Zeitdauer). Wie in der VO erwähnt: *"Die Simulationszeit springt von Ereigniszeitpunkt zu Ereigniszeitpunkt und nur dabei wird die Simulationsuhr weiter gesetzt."*
- Prozessorientierter Modellierungsstil: Der Scheduler steuert im Hintergrund die Abarbeitung beliebig vieler Prozesse. Die Prozesse werden in richtiger Reihenfolge und den dafür vorgesehen Zeitabständen vom Scheduler ausgeführt. Wobei der Scheduler auch für die Verwaltung und die (Re-)Aktivierung der jeweiligen Prozesse zuständig ist. Zeitverbrauchende Aktivitäten werden mittels inaktiver Prozessphasen abgebildet:
  - hold(): Abbildung einer zeitverbrauchenden Aktivität. Inaktiv für ein vorgegebenes fixes Intervall der Simulationszeit (Kontrolle an den Scheduler)
  - passivate(): Abbildung eines Wartezustands. Inaktiv für unbestimmte Dauer der Simulationszeit (Kontrolle auf unbestimmte Zeit abgegeben, Reaktivierung meist durch andere Prozesse)

```
java.lang.Object
└ desmoj.core.simulator.NamedObject
  └ desmoj.core.simulator.Scheduler
```

In Desmoj ist der Scheduler mittels einer eigenen Klasse dargestellt und der Konstruktor ist wie folgt dargestellt:
Scheduler(Experiment exp, java.lang.String name, EventList eventList)
Erzeugt einen Scheduler mit dem gegebenen Namen und einer Event-Liste.

2.5 Experiment

```
java.lang.Object
└ desmoj.core.simulator.NamedObject
  └ desmoj.core.simulator.Experiment
```

Hier folgen die wichtigsten Methoden betreffend Simulationszeit:

ShowProgressBar

- *public boolean isShowProgressBar():* liefert einen boolean Wert wahr wenn ein Verlaufsbalken für dieses Experiment aktiviert wäre.
- *public void setShowProgressBar(boolean newShowProgressBar):* diese Funktion muss aufgerufen werden, bevor die Simulation startet. Mit Parameter true, wird für dieses Experiment ein Verlaufsbalken angezeigt. Mit false, wird kein Balken angezeigt.

- *public void setShowProgressBarAutoclose(boolean autoclose):* hat nur Auswirkungen, wenn zuvor bei setShowProgressBar(true) ausgewählt wurde. Muss ebenfalls vor Beginn des Experiments ausgeführt werden. Stellt ein ob der Verlaufsbalken nach Ablauf des Experiments automatisch geschlossen werden soll. Parameter: true, der Balken schließt sich nach Ablauf des Experiments. false, der Balken bleibt offen, bis dieser vom User manuell geschlossen wird.

tracePeriod

- *public void tracePeriod(TimeInstant startTime, TimeInstant stopTime):* Hier kann ein gewisser Zeitraum gewählt werden, in welchem ein Simulations Output generiert werden soll (z.B. .csv Dateien). Teilweise können Simulationen eine gewisse Vorlaufzeit benötigen, bis Werte entstehen welche für das Experiment tatsächlich von Relevanz sind. Mit dieser Funktionen kann hierfür eine Zeitspanne gewählt werden. Optional kann auch im jeweiligen Programmcode mit *public void traceOn(TimeInstant startTime), public void traceOff(TimeInstant stopTime)* die Output-Erstellung gesteuert werden.

start

- *public void start():* startet das Experiment mit default Zeit = 0. Kann nur einmal in einem Experiment verwendet werden. Es initialisiert das verknüpfte Modell und startet die Simulation.
- *public void start(TimeInstant initTime):* startet das Experiment mit der angegebenen Simulationszeit als Startzeit. Allerdings wird dieses erst wirklich gestartet, sobald ein gültiges Modell damit verknüpft wurde.

stop

- *public void stop(ModelCondition stopCond):* damit können auch mehrere (durch mehrmaligen aufrufen mit unterschiedlichen Bedingungen) Modellbedingungen gesetzt werden, unter welchen die Simulation gestoppt werden soll. Also z.B. ab einem erreichten Wert. Sobald eine der Bedingungen erfüllt ist wird die Simulation gestoppt. Um zu vermeiden, dass die Simulation gegebenenfalls unendlich weiterläuft empfiehlt es sich trotzdem die Simulation mit dem Setzen einer vorgegeben Zeit zu stoppen. Wie in der nächsten Methode beschrieben.
- *public void stop(TimeInstant stopTime):* stoppt die Simulation an einer gegebenen Simulationszeit. Falls keine gültige Zeit als Parameter übergeben wurde, gilt 0 als default Wert und die Simulation kann nicht weiter als bis zu diesem Zeitpunkt laufen. Mehrmaliges aufrufen dieser Methode überschreibt die zuvor festgesetzte Zeit. public void stop(): stoppt die Simulation sofort zur aktuellen Simulationszeit. Diese kann aber später mit proceed() wieder fortgesetzt werden.

ExecutionSpeedRate

- *public void setExecutionSpeedRate(double rate):* damit kann die Geschwindigkeitsrate der Durchführung gesteuert werden. Wird der Wert größer als 0 gesetzt verhält sich das proportional zur realen Uhrzeit.
  Folgende Gleichung gilt hier für eine Geschwindigkeitsrate > 0 :
  rate $*$ simulation-time = wall-clock-time (reale Uhrzeit).
  Falls die Rate = 0 oder < 0 ist: die Simulation wird so schnell wie möglich durchgeführt. Der default Wert beträgt = 0 (also so schnell wie möglich).

get-Methoden

- *getSimClock():* liefert die eingestellte SimClock für dieses Experiment. ModelComponents benötigt Zugriff zur SimClock um die aktuelle Simulationszeit abzurufen.
- *getModel():* liefert das Modell das aktuell mit dem Experiment verbunden ist, bzw. den Wert NULL falls noch kein Modell eingestellt wurde.
- *getScheduler():* liefert den eingestellten Scheduler für dieses Experiment.
- *getRealTimeStartTime():* liefert die echte Startzeit als long Wert.
- *getStopTime():* liefer die TimeInstant an welcher erwartet wird, dass das Experiment stoppt.
- *getExecutionSpeedRate():* liefert die aktuelle Geschwindigkeitsrate als double Wert.

Literatur

1. `http://desmoj.sourceforge.net/tutorial/overview/0.html`
2. `http://desmoj.sourceforge.net/doc/desmoj/core/simulator/Experiment.html`