Anmerkungen zur Seminararbeit

Aufgabenstellung

In der Aufgabenstellung heißt es:

Die Anwendung muss nicht verteilt lauffähig sein. Es genügt, wenn die Clients und die Verwaltungskomponente gemeinsam auf einem Rechner im selben Prozess ablaufen.

Dieser Hinweis war eigentlich als Vereinfachung für diejenigen Gruppen gedacht, die nicht mit dem Aktorensystem "Thespian" arbeiten möchten und eine eigene Implementierung der Aktoren verwenden (wichtig: jeder Aktor kann immer nur von einem Thread durchlaufen werden).

Wenn die Verwaltungskomponente und die Clients in einem gemeinsamen Prozess laufen sollen, ist die Implementierung des User-Interface etwas umständlich: Zunächst würde das Aktorensystem gestartet und dann könnte bspw. in einer Endlos-Schleife wahlweise immer einer der Clients (Kunden bzw. Verwaltung) zum Zuge kommen und mit dem Aktorensystem sprechen. Das ist allerdings ziemlich künstlich (und der Zugriff auf das Aktorensystem erfolgt nicht parallel).

In einer realistischeren Implementierung würde jeder Client in einem eigenen Prozess laufen und hat sein eigenes User-Interface. Wenn Sie dies mit einer "selbstgestrickten" Implementierung der Aktoren in Python realisieren möchten, ist das unter Verwendung des Moduls `multiprocessing` möglich:

https://docs.python.org/3/library/multiprocessing.html

Mit Thespian ist das aus Sicht des Dozenten allerdings einfacher umsetzbar, siehe dazu den folgenden Abschnitt.

Hinweise zur Implementierung mit Thespian

Wenn Sie mit Thespian arbeiten, ist die Implementierung des Aktorensystems und der Clients in getrennten Prozessen zu bevorzugen und auch recht einfach zu realisieren.

Das Aktoren-System muss nur mit der Option `multiprocTCPBase` gestartet werden. Am Code der Aktoren ändert sich nichts. Schematisch würde das so aussehen:

```
system = ActorSystem("multiprocTCPBase")
system.listen()
```

Jetzt wartet das Aktoren-System auf den Eingang von Nachrichten.

Der Prozess mit dem Aktoren-System wird gestartet (d. h. der Interpreter mit dem Python-Skript als Parameter wird auf der Kommandozeile aufrufen).

Der Code für die Kunden- bzw. den Verwaltungs-Client wird jeweils in einem eigenen Modul implementiert und in einem eigenen Prozess gestartet (wie das Aktoren-System, s. o.). Die Clients müssen nur eine Verbindung zum laufenden Aktoren-System herstellen:

```
system = ActorSystem("multiprocTCPBase")
```

Nun können die Clients Aktoren erzeugen. Hier wird ein Aktor der Klasse `AdminActor` erzeugt, der im Modul ticket_store.py implementiert ist:

```
admin = system.createActor("ticket_store.AdminActor")
```

So würde bspw. eine synchrone Nachricht an den Aktor versendet (hier ist die Nachricht ein Objekt der Klasse `ListEventsMessage`, offensichtlich kommt eine Liste zurück):

```
event_list = system.ask(admin, ListEventsMessage())
```

Es ist üblich, wie hier gezeigt, eigene Klassen für die Nachrichten zu erstellen. Sie können aber auch eingebaute Datentypen, bspw. Strings, als Nachrichten verwenden.

Hilfreich ist die Dokumentation zu Thespian, insbesondere die Beschreibung der Schnittstellen von `ActorSystem` und `Actor`:

https://thespianpy.com/doc/using.html

Anmerkung: Erzeugung des Aktoren-Systems

Das Aktoren-System in Thespian ist ein Singleton: Der erste Aufruf von `ActorSystem()` bestimmt die Konfiguration des Systems. Alle weiteren Aufrufe von `ActorSystem()` liefern immer eine Referenz auf das beim ersten Aufruf erzeugte System. Im oben beschriebenen Beispiel wird dieses System beim Start des Prozesses mit dem Aktoren-System erzeugt. Die darauf folgenden Aufrufe der Client-Prozesse verwenden dann dieses System.