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 Abschnitt zur Implementierung weiter unten.

Anforderungen

Die Daten können während der Laufzeit der Anwendung im Speicher gehalten werden, d. h. Sie müssen keine Datenbank einsetzen (natürlich dürfen Sie mit einer Datenbank arbeiten, das kann die Implementierung evtl. sogar vereinfachen). Empfehlenswert bei einer Implementierung in Python ist Sqlite:

https://docs.python.org/3/library/sqlite3.html

Wenn (wie empfohlen, s. u.) Ihre Aktoren jeweils in einem eigenen Prozess laufen, erleichtert das die Implementierung erheblich (Prozess-übergreifender Zugriff auf die Daten).

Das Jahresbudget des Kunden kann wahlweise auf Client-Seite oder in der Verwaltungskomponente des Systems verwaltet werden. Die Idee ist, dass der Kunde solange Tickets für Veranstaltungen, die in einem Kalenderjahr liegen, buchen kann, bis sein Jahresbudget erschöpft ist. Beispiel: Der Kunde möchte eine Veranstaltung buchen, die in 2020 stattfindet. Das System gibt direkt vor der Bestellung aus, wieviel Budget in 2020 bereits ausgegeben wurde (was recht einfach aus den vorhandenen Daten berechnet werden kann).

Eine Authentifizierung (d. h. ein Login) ist nicht gefordert, auch nicht für den Administrations-Client. Kunden können bspw. an Hand Ihres Namens oder einer ID identifiziert werden. Eine Registrierung ist nicht notwendig, d. h. jeder Kunde kann einfach bestellen; es ist ja auch keine Bezahlung vorgesehen :-)

Die Teilnehmeranzahl pro Veranstaltung (d. h. die Maximalanzahl von Tickets) kann je nach Veranstaltung variieren. Sie kann beim Anlegen der Veranstaltung über den Administrations-Client angegeben werden.

Folgende Anforderung an den Verwaltungs-Client war nicht eindeutig formuliert: "Die Anzahl der verkauften Tickets pro Veranstaltung kann abgerufen werden." Damit ist gemeint, dass eine Liste der Veranstaltungen mit den jeweils bis zu diesem Zeitpunkt verkauften Tickets abgebucht werden kann.

Ein Login-Mechanismus für die Anmeldung an das System ist nicht gefordert.

Sie dürfen den Client für den Kunden bzw. für die Verwaltung auch als ein gemeinsames Programm implementieren, das im Kunden- bzw. Admin-Modus gestartet wird.

Sie können von einem "gutwilligen" Nutzer ausgehen, d. h. es müssen keine aufwändigen Fehlerprüfungen (bspw. hinsichtlich des Datumsformats) implementiert werden.

Sie können einfache Datenstrukturen bzw. Datentypen für die Speicherung der Daten verwenden. Bspw. kann die Adresse eines Kunden komplett als String gespeichert werden.

Hinweise zum Design

Die Aktoren sollen in diesem System zwei Aufgaben erfüllen:

- Nebenläufige Verarbeitung von Anfragen, wo dies gefahrlos möglich ist. Beispiele sind Anfragen eines Kunden nach der Liste von Veranstaltungen oder den aktuell gebuchten Tickets.
- Serialisierung von Anfragen, die einen kritischen Abschnitt enthalten, also bspw. die Buchung von Tickets.

Daraus folgt, dass jeder anfragende Client von einem eigenen Aktor "bedient" wird, der Zugriff auf kritische Abschnitte aber nur von einem einzigen Aktor durchgeführt wird. Oder anders ausgedrückt: alle Zugriffe auf kritische Abschnitte müssen durch den "Flaschenhals" eines einzigen Aktors hindurch (s. auch den folgenden Abschnitt zur Implementierung mit Thespian).

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

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.

Erzeugung von Aktoren

Normalerweise wird beim Aufruf von `.createActor()` (s. o.) jeweils ein neuer Aktor (d. h. ein neues Objekt der entsprechenden Klasse) erzeugt.

Es gibt auch die Möglichkeit, einen *Named Actor* zu erzeugen, der dann bei weiteren Aufrufen von `.createActor()` mit dem vergebenen Namen nicht neu erzeugt wird. Stattdessen wird die Referenz auf den existierenden Aktor zurückgegeben (dieser Aktor ist also ein Singleton). Siehe dazu die Dokumentation:

https://ogy.de/9bbs

Aktoren können nicht nur über das `ActorSystem`, sondern auch in anderen Aktoren erzeugt bzw. referenziert werden. Siehe dazu die Dokumentation:

https://ogy.de/t0w2