Universität Bremen

Reaktive Programmierung SoSe 17

Christoph Lüth
Martin Ring

2. Übungsblatt

Ausgabe: 27.04.17
Abgabe: 11.05.17

2.1 *Promise: My Future in Haskell* — *3 Punkte*

In dieser Aufgabe wollen wir Futures und Promises wie wir sie in Scala kennengelernt haben in Haskell implementieren. Dazu implementieren wir ein Modul Future mit den folgenden Funktionen:

```
data Promise α
data Future α

promise   :: IO (Promise α)                  -- Erzeugt ein Promise
complete  :: Promise α → α → IO ()           -- Erfüllt ein Promise

future    :: Promise α → Future α            -- Die Future zu einem Promise
onComplete :: Future α → (α → IO ()) → Future α
wait      :: Future α → IO α                 -- Wartet, dass die Future erfüllt wird
```

Dabei sind Future und Promise komplett asynchron; es wird nirgendwo ein neuer Thread erzeugt.

Das Beispiel Robots.hs (in der Vorlage enthalten) zeigt die Verwendung von Future und Promise mit der oben angegebenen Schnittstelle; es sollte sich bei korrekter Implementierung wie das Scala-Beispiel aus der Vorlesung verhalten.

2.2 *MVar in Scala?!* — *2 Punkte*

Nachdem wir Future in Haskell implementiert haben, wollen wir den anderen Weg gehen und MVar in Scala implementieren. Oder auch nicht — was ist das große Problem bei einer Implementation von MVar in Scala?

Skizzieren Sie eine Lösung (welche Schnittstelle hätte eine Scala-Implementierung), und schildern Sie die auftretenden Probleme.

2.3 *6 Nimmt!* — *15 Punkte*

Jetzt wollen wir die ganzen Futures, Promises und MVars endlich auch mal zur Entspannung nutzen. Das Kartenspiel 6 Nimmt! der deutschen AMIGO GmbH ist zwar nicht ganz frei von Copyright, für den Privatgebrauch ist gegen eine digitale Nachbildung jedoch sicherlich außer eine geringen fünfstelligen Abmahngebühr kaum etwas einzuwenden.

Ihre Aufgabe ist es, einen Spielserver zu implementieren, mit dem sich bis zu 10 Spieler verbinden können um gegeneinander 6 Nimmt! zu spielen. Die Spielregeln sind unter `https://www.amigo-spiele.de/spiel/6-nimmt` in verschiedenen Sprachen verfügbar (Die Profi-Variante können Sie ignorieren).

Diese Aufgabe kann entweder in Scala (mit Promises und Futures) oder in Haskell (mit MVars) gelöst werden.

- Modellieren Sie geeignete Datenstrukturen für den Spielzustand eines 6 Nimmt! Spiels:
  - Spieler sollten zusätzlich zu Hand- und Minuskarten über einen Namen identifiziert werden;
  - Karten haben neben dem Wert (1-104) auch Minuspunkte (siehe Anleitung);
  - auf dem Tisch liegen immer vier Kartenreihen (1-5 Karten lang).
- Implementieren Sie nun die eigentlichen Spielzüge:

  - Alle Spieler wählen nebenläufig eine ihrer Handkarten aus (Future/MVar).
  - Wenn jeder Spieler gewählt hat, werden die Karten aufsteigend nach Wert in die Kartenreihen einsortiert.
  - Wenn eine Karte niedriger ist als die kleinste obere Karte auf dem Tisch, muss der Spieler, welcher diese Karte ausgewählt hat, befragt werden welche Kartenreihe er tauschen möchte (Future/MVar).
- Stellen Sie eine Schnittstelle bereit, um sich mit dem Spielserver zu verbinden. Ihnen steht frei, entweder wie in Übung 1 einen Webserver zu implementieren, oder einen einfachen Socket bereit zu stellen (java.net.ServerSocket/Network.Socket).

Zu guter Letzt brauchen wir noch einen geeigneten Client. Implementieren Sie dafür abhängig von ihrer Wahl der Servertechnologie entweder einen Webclient oder einen einfachen kommandozeilenbasierten Client, der sich mit dem Socket verbindet. Der Spielablauf aus Clientsicht soll so aussehen:

1. Beim Verbindungsaufbau gibt der Spieler seinen Namen an
2. Der Spieler wird über die anderen verbundenen Spieler informiert (wird aktualisiert)
3. Wenn alle Spieler gemeinsam entschieden haben, dass das Spiel beginnen kann, fängt die erste Runde an
4. Der Spieler bekommt seine Handkarten und die Tischkarten angezeigt
5. Der Spieler kann nun eine seiner Handkarten auswählen
6. Wenn alle Spieler eine Karte gewählt haben, wird angezeigt, wie die Karten einsortiert werden
7. Muss ein Stapel zum Tauschen ausgewählt werden, wird das dem entsprechenden Spieler angezeigt
8. Wenn alle Karten einsortiert wurden und noch Handkarten vorhanden sind beginnt ein neuer Zug (4.)
9. Ansonsten ist die Runde beendet.
10. Wenn keiner der Spieler mehr als 66 Minuspunkte gesammelt hat, wird eine neue Runde begonnen (4.)
11. Ansonsten ist das Spiel beendet.
12. Der Gewinner ist der Spieler mit den wenigsten Minuspunkten.

Viel Spaß beim Spielen!

Hinweis:

Wieviel auf dem Client und wieviel auf dem Server passiert, und wie diese miteinander kommunizieren ist Ihnen freigestellt. Dokumentieren Sie ihre Entscheidungen.