Universität Bremen

Reaktive Programmierung SoSe 15

Christoph Lüth
Martin Ring

4. Übungsblatt

Ausgabe: 11.06.15
Abgabe: 25.06.15

Dieses Aufgabenblatt ist die Zusammenführung der beiden vorherigen Übungsblätter: auf Insistieren der Vertriebsabteilung ist die grafische Benutzerschnittstelle zurück, aber diesmal für das aktorenbasierte System.

4.1 *GUI Actor GUI!* — *15 Punkte*

Implementieren Sie unter Nutzung Ihrer Lösung aus Übungsblatt 2 eine grafische Benutzerschnittstelle für die Robo-Mine.

Konkret bedeutet dies, dass es einen Aktor gibt, der einen Roboter in der Mine modelliert, dessen Zustand auf der grafischen Benutzerschnittstelle visualisiert wird, und dessen Aktionen von dem Benutzer gesteuert werden.

Die Implementierung besteht aus zwei Teilen: den *Client*, der mit *Scala.js* nach JavaScript übersetzt wird und im Web-Browser läuft, und dem *Server*. Beide müssen getrennt übersetzt werden (der Client nach JavaScript, der Server normal nach Bytecode). Es sollen sich mehrere Clients mit einem Server verbinden können.

Hinweis: Für die Kommunikation zwischen Client und Server empfehlen wir die Nutzung von *akka-http*[1], einer Bücherei, welche die Kommunikation mit Aktoren via HTTP erlaubt. Auf dem Server kann damit ein Aktor eine HTTP-Schnittstelle anbieten, mit welcher sich ein Browser verbinden kann. Der Browser kann dann den eigentlichen Client herunterladen. Die eigentliche Kommunikation zwischen Server und Client erfolgt über WebSockets.

Auf der Webseite finden Sie eine Beispielapplikation, welche dieses Prinzip in Aktion zeigt. Die Beispielapplikation zeigt auch, wie Sie *sbt* so konfigurieren, dass es wie oben beschrieben den Client mit *Scala.js* und den Server wie üblich übersetzt.

4.2 *Automatisierung* — *5 Punkte*

Die Bedienung der Mine ist auf die Dauer etwas eintönig. Daher wäre es wünschenswert, den Client skripten zu können, d.h. kleine Programme in einer einfachen Programmiersprache (“Skriptsprache”) zu schreiben, die einfache Aufgaben (beispielsweise “fahre solange gerade aus, bis du auf ein Hindernis triffst” oder “fahre zum nächsten Gold”) erledigen.

Dazu brauchen wir:

- Eine einfache Programmiersprache mit Kontrollkonstrukten (Iteration und Fallunterscheidung), Basisanweisungen (entspricht den möglichen Kommandos: in eine Richtung fahren, aufnehmen, pausieren), und Eingaben (entspricht den Sensoren und dem Zustand des Roboters: Energielevel, Gold, Radar, Scanner), sowie Variablen und grundlegenden arithmetischen Operationen.
- Skripte bestehen dann aus Definitionen von Prozeduren, und Anweisungen, welche Prozeduren an bestimmte Tasten binden. Eine mögliche Erweiterung wäre, Skripte automatisch (jeden Kontrollturnus) oder unter bestimmten Bedingungen (bspw. Energie wird zu niedrig) laufen zu lassen.
- Der Client benötigt einen Parser, der das Skript (lokal) liest und interpretiert. Für die Entwicklung des Parsers gibt es in der Scala-Standardbücherei Parserkombinatoren[2].

Eine weniger ambitionierte Lösung der Aufgabe besteht in der Entwicklung einer plugin-Architektur, welche die Erweiterung des Clients in Scala erlaubt. Ein Plugin implementiert dann eine zu definierende Schnittstelle; die Installation neuer Plugins erfordert das System neu zu übersetzen.

[1] `http://doc.akka.io/docs/akka-stream-and-http-experimental/1.0-M2/scala/http/`
[2] `scala.util.parsing.combinators`