Hochschule Rosenheim, Fakultät für Informatik

Vorlesung: Verteilte Verarbeitung SS 2017 Prof. Dr. Gerd Beneken

Übungsblatt 1 von 4 – Praktikum als Grundlage für ein Kolloquium

Ziel dieses Praktikums

Dieses Praktikum hat zwei Ziele, erstens sollen Sie nach erfolgreichem Abschluss aller Praktikumsaufgaben verteilte Systeme besser verstehen und diese praktisch umsetzen können. Zweitens sollen Sie möglichst viel praktische Erfahrungen beim Programmieren sammeln, damit sie das Praxissemester möglichst erfolgreich überstehen.

Ziel ist es, dass Sie während des Programmierens möglichst umfangreich Feedback zu Ihren Ergebnissen erhalten und zwar von dem betreuenden Professor und auch von Ihren KommilitonInnen. Je mehr Feedback Sie erhalten, desto mehr und besser lernen Sie. Ich hoffe, dass dadurch auch Ihre Motivation steigt, sich in das Thema Programmierung zu vertiefen.

Alle Aufgaben sind jeweils über einen Zeitraum von etwa einem Monat zu bearbeiten. Das Semester wird in vier große Praktika unterteilt. Diese sind jeweils im Laufe des Semesters (weit vor dem Termin des Kolloquiums abzugeben).

Ziel der Aufgabe: Reaktive Systeme

Wenn Sie diese Aufgabe erfolgreich abgeschlossen haben, können Sie

- Dateizugriff in Java und anderen Sprachen über Streams und Sie haben das Konzept der Streams verstanden
- Umwandeln von Java Objekten in JSON-Strings sowie in XML-Strings und Sie haben grob verstanden wie JSON und XML sowie XML-Schema funktionieren.
- Programmierung eines endlichen Automaten (wir bauen eine Mealy-Maschine) in einer beliebigen Programmiersprache.
- Bau eines einfachen reaktiven Systems und sie haben verstanden, was ein reaktives System ist.
- Konzepte der Synchronen und Asynchronen Verarbeitung: Polling, Producer / Consumer, Executor-Framework und Futures.

Der Aufgabentext wird Ihnen völlig unlösbar erscheinen (das ist Absicht). Die Kunst bestseht jetzt darin, dieses **viel zu komplizierte Problem** in für Sie machbare Teilschritte zu zerlegen und diese dann, soweit Sie kommen, abzuarbeiten. Also von einem kleinen Teilprogramm zum nächsten und diese dann wieder zu einer Gesamtlösung zu integrieren.

Aufgabe:

Mealy-Maschine: Implementieren Sie eine Mealy-Maschine in Java. Diese Maschine hat

- Zustände (z.B. modelliert über Objekte einer Klasse State).
- Eingabealphabet (z.B. über Objekte einer Klasse Symbol)
- Ausgabealphabet (z.B. über Objekte einer Klasse Symbol), Eingabe- und Ausgabealphabet sind also identisch.
- Zustandsübergangsfunktion, diese bekommt als Parameter ein State-Objekt, ein Input-Symbol-Objekt und gibt ein Objekt der Klasse State zurück. Die Zustandsübergangsfunktion verwendet im Hintergrund bitte eine Zustandsübergangstabelle (als Indizes verwenden Sie State-Objekte und InputSymbol Objekte; in den Zellen der Tabelle stehen dann State-Objekte jeweils als Folgezustand).

- Ausgabefunktion, diese bekommt als Parameter ein State-Objekt und ein Input-Symbol-Objekt und gibt ein Output-Symbol Objekt zurück. Die Ausgabefunktion verwendet bitte auch eine Tabelle wieder mit den Indizes State-Objekt und Input-Symbol-Objekt, in den Zellen der Tabelle stehen Funktionen (Lambda-Expression, Command oder ähnliches), welche ihrerseits ein InputSymbol Objekt und ein State-Objekt erhalten und ein Output-Symbol-Objekt zurückgeben.

Laden der Maschine aus XML-Datei: Programmieren Sie eine Funktionalität mit der Sie die oben genannte Mealy-Maschine aus einer XML-Datei laden kann. Damit könnten Sie sich eigene / neue Automaten überlegen, ohne dass Sie noch den Java-Code ändern müssten. Die Datei muss alle Informationen zur Mealy-Maschine enthalten: Zustände, Eingabe- und Ausgabe-Alphabet sowie die Zustandsübergangs- und die Ausgabefunktion.

Symbole zunächst als einfache Buchstaben: Programmieren Sie die erste Version Ihrer Maschine derart, dass Eingabe- und Ausgabealphabete jeweils Buchstaben sind, wie sie es vermutlich in Grundlagen der Informatik gesehen haben. Die Maschine konsumiert also eine Zeichenkette und produziert daraus eine andere Zeichenkette. Eventuell können Sie für die ersten Tests auch (Mealy-)Automaten aus GDI1 verwenden. Die Buchstaben kann Ihre Maschine ja von der Konsole einlesen und die Ausgabe in eine Datei machen.

Symbole als JSON Strings: Als Symbole verwenden Sie nun bitte JSON Strings. Diese stellen Nachrichten dar. Die Nachricht hat einen Typ (Request, Response, Fault, Ackowledge, Ping, …) sowie einen Payload in Forme irgendeines Strings. Der Nachrichtentyp wird von unserem Automaten als Element des Eingabe- und des Ausgabealphabetes betrachtet. Damit werden aus den JSON-Strings nach dem Parsen Objekte des Typs InputSymbol.

Nachrichten aus Dateien lesen: Jeder JSON-String ist in einer eigenen beliebig benannten Datei mit der Endung .msg gespeichert. Diese Dateien befinden sich im Verzeichnis input. Ihr Programm soll dieses Verzeichnis überwachen und jede Datei lesen und danach entferen. Jede Datei wird damit in ein Objekt vom Typ InputSymbol übersetzt.

Ausgabe in Dateien: Auch die Ausgabe soll in Form derselben JSON-Strings (Request, Response etc.) erfolgen. Damit erzeugt Ihr Automat Objekte vom Typ Symbol (eventuell können OutputSymbol und INputSymbol über dieselbe Klasse dargestellt werden). Jedes Output-Symbol Objekt wird in ein Verzeichnis output geschrieben, jeweils als eigene Datei, welche die ausgabe wieder als JSON-String enthält.

Polling: Bauen Sie Ihre Software nun so um, dass ein eigener Thread das input-Verzeichnis überwacht. Wenn eine neue Datei in diesem Vereichnis auftaucht liest der Thread die Datei (InputSymbol) und schreibt den gelesenen JSON-String in eine Queue. Ein weiterer Thread liest aus der Queue die gelesenen JSON-Strings und füttert damit die Mealy-Maschine.

Asynchrone Verarbeitung: Die Ausgabefunktionen der Mealy-Maschine soll asynchron über das Executor-Framework oder über Futures geschehen. Erzeugen Sie für jede Ausgabe (InputSymbol x Zustand -> OutputSymbol) ein Runnable / Callable und übergeben Sie dieses dem Executor Framework zur asynchronen Ausführung.