Hochschule Rosenheim, Fakultät für Informatik

Vorlesung: Verteilte Verarbeitung SS 2018 — Prof. Dr. Gerd Beneken
Nachholprüfung WS 18/19 — Lehrbeauftragter Thomas Mildner

Übungsblatt 1 von 3 – 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 viele 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 besteht 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 Input Symbol 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 die Output-Symbole.

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 von der Standard-Eingabe und produziert daraus eine andere Zeichenkette auf der Standard-Ausgae. Eventuell können Sie für die ersten Tests auch (Mealy-)Automaten aus GDI1 verwenden

Nachrichten aus Dateien lesen: Als Symbole verwenden wir Dateien in denen bespielsweise ein JSON String gespeichert sein kann. Als Input-Symbol dient dabei der Dateiname mit der Endung .msg gespeichert. Diese Dateien befinden sich im Verzeichnis input. Ihr Programm soll dieses Verzeichnis überwachen und jede Datei lesen und danach entfernen. Jede Datei wird damit in ein Objekt vom Typ Input Symbol übersetzt. Beispielsweise wird die Datei request.msg zum Symbol REQUEST oder die fault.msg zu FAULT. Ziel ist hier, dass wir mit unserer Mealy-Maschine einen einfachen Protokoll-Automaten bauen.

Ausgabe in Dateien: Auch die Ausgabe soll in Form von (leeren) Dateien erfolgen. Damit erzeugt Ihr Automat Objekte vom Typ Output-Symbol, das Symbol wird wieder nur über den Dateinamen identifiziert, beispielsweise reply.msg oder ackowledge.msg. Jedes Output-Symbol Objekt wird in ein Verzeichnis output geschrieben, jeweils als eigene Datei (klar wegen des Dateinamens).

Polling: Bauen Sie Ihre Software nun so um, dass ein eigener Thread das input-Verzeichnis überwacht. Wenn eine neue Datei in diesem Verzeichnis auftaucht liest der Thread die Datei (Input Symbol) 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.

Thread-Kommunikation mit Queues: Die Mealy-Maschine erhält alle Eingabe (Sprich die Nachrichten bzw. die Symbole des Eingabealphabets) über eine **Blocking-Queue**. Die Mealy-Maschine wird in einem eigenen Thread ausgeführt. Sie liest die Symbole aus der Blocking- Queue, führt daraufhin den Zustandsübergang durch. Die Ausgabe der Mealy-Maschine erfolgt in eine zweite Blocking Queue. Damit hat die Maschine zwei Queues, eine für die eingehenden Nachrichten / Symbole und eine für die erzeugten Nachrichten / Symbole. Lagern Sie das Überwachen des Verzeichnisses und das Lesen der Eingabesymbole aus dem Verzeichnis „input“ sowie as Schreiben der Ausgabesymbole in das Verzeichnis „output“ jeweils in einen eigenen Thread aus. Die Kommunikation dieser Threads mit der Mealy-Maschine erfolgt jeweils über die bereits genannten Queues.

Tipps:

- Starten Sie zuerst mit einem definierten Automaten ohne Einlesen einer XML Datei. So kann die Funktionalität der Übergangsfunktion und vom Automaten sichergestellt werden.
- Passende .gitignore anlegen
- Nach dem erfolgreichen Test des Automaten, startet den XML Import.
- Überlegt euch sorgfältig, wie die Anwendung strukturiert werden kann. Welche Klasse / welches Modul ist für welche Aufgabe zuständig? Wie könnte die Anwendung sinnvoll strukturiert werden?
- Arbeitet mit Packages und achtet auf eine saubere Struktur und eine gute Code Qualität!