Projektarbeit

Thema:

Gemeinsamkeiten und Unterschiede von Service-orientierte Architektur (SOA) und Microservices

Stefan Kruk

geboren am 14.08.1992

Matr.-Nr.: 7084972

An der Fachhochschule Dortmund im Fachbereich Informatik erstellte Projektarbeit

im Studiengang Softwaretechnik (Dual) - Modul Entwicklung verteilter Anwendungen

Betreuer: Prof. Dr. Johannes Ecke-Schüth

Fachhochschule Dortmund

University of Applied Sciences and Arts

Fachbereich Informatik

Dortmund, 19. Januar 2016

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich die vorliegende Arbeit selbständig angefertigt und mich keiner fremden Hilfe bedient sowie keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichten oder nicht veröffentlichten Schriften und anderen Quellen entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Dortmund, den 19. Januar 2016 Stefan Kruk

Erklärung

Mir ist bekannt, dass nach § 156 StGB bzw. § 163 StGB eine falsche Versicherung an Eides Statt bzw. eine fahrlässige falsche Versicherung an Eides Statt mit Freiheitsstrafe bis zu drei Jahren bzw. bis zu einem Jahr oder mit Geldstrafe bestraft werden kann.

Dortmund, den 19. Januar 2016 Stefan Kruk

Inhaltsverzeichnis

Überblick

Kurzfassung

In dieser Arbeit sollen die Vor- und Nachteile von Service-orientierte Architektur und Microservices erörtert und anschließend miteinander verglichen werden. Dabei soll explizit auf die Unterschiede und Gemeinsamkeiten der beiden Architekturmodelle eingegangen werden. Für eine bessere Verständlichkeit wird zusätzlich ein bestimmter Prozesskontext mit beiden Modellen implementiert.

Abstract

In this Projekt i will describe the Pro and Cons of service-oriented architecture and Microservices. After that i will compare both architectural models and describe the differences and similarities between these two. For a better understanding i will implement an application with both architectural models.

Kapitel 1

Einleitung

1.1 Motivation

In einem Unternehmen sind oft verschiedene Softwareprodukte im Einsatz. Damit ein reibungsloser Ablauf der internen Prozesse gewährleistet werden kann, sollte im Idealfall Software Interoperabel sein und so ohne weitere Probleme mit einander Arbeiten. Eine Lösung ist die Service-orientierte Architektur. Dabei wird ein ganzer Geschäftsprozess in einer Monolithischen Anwendung, welcher zu einem bestimmten Kontext gehört, abgebildet. Eine andere Lösung sind Microservices. Diese Services sind, wie der Name schon sagt, in kleine und selbständige Anwendungen aufgeteilt, die nur eine bestimmte Aufgabe haben. Außerdem enthalten sie nur die reine Businesslogik. Damit Microservices für einen bestimmten Kontext genutzt werden können, müssen sie zusätzlich Orchestriert werden.

In dieser Arbeit sollen die Vor- und Nachteile beider Lösungen, unter anderem durch eine Implementierung, erörtert und danach miteinander verglichen werden. Abschließend werden die Ergebnisse präsentiert.

1.2 Ausgangssituation

1.3 Vorgehen

Kapitel 2

Theorie neuronaler Netze

2.1 Die formalen Neuronen

In diesen Abschnitten geht es um die mathematischen Grundlagen der zu behandelnden Problemstellung. Hier können Definitionen und Sätze auftauchen, wobei die Sätze teils bewiesen werden sollten (falls einfach und für das weitere Verständnis wesentlich) oder ihre Beweise sauber zitiert werden. Beispiele:

Definition 2.1.1 (Formales Neuron)
Ein formales Neuron ist eine Funktion $\kappa : \mathbf{R}^n \to \mathbf{R}^m$*, die erzeugt wird durch die Verkettung einer sogenannten Transferfunktion* $\sigma : \mathbf{R} \to \mathbf{R}$ *mit einer sogenannten Aktivierungsfunktion* $A : \mathbf{R}^n \to \mathbf{R}$*:*

$$\begin{aligned} \kappa : \quad & \mathbf{R}^n \to \mathbf{R}^m \,, \\ & \vec{x} \mapsto \big(\sigma(A(\vec{x})), \sigma(A(\vec{x})), \,\ldots\,, \sigma(A(\vec{x}))\big) \,. \end{aligned} \tag{2.1}$$

Satz 2.1.1 (Dichtheitssatz für hyperbolische Aktivierung)
Es sei $f : \mathbf{R}^n \to \mathbf{R}$ *stetig,* $K \subset \mathbf{R}^n$ *kompakt und* $\sigma : \mathbf{R} \to \mathbf{R}$ *eine stetige Sigmoidalfunktion. Dann gibt es für alle* $\varepsilon > 0$ *Parameter*

$$\begin{aligned} & q \in \mathbf{N}, \\ & d_{kp} \in \mathbf{R}, \;\; 1 \le k \le n, \; 1 \le p \le q, \\ & \rho_p \in \mathbf{R}, \quad 1 \le p \le q, \\ & g_p \in \mathbf{R}, \quad 1 \le p \le q, \end{aligned} \tag{2.2}$$

so dass für alle $\vec{x} \in K$ gilt

$$\left| f(\vec{x}) - \sum_{p=1}^{q} g_p \sigma \left(\rho_p \prod_{k=1}^{n} (x_k - d_{kp}) \right) \right| < \varepsilon. \tag{2.3}$$

Definiert man nun

$$\sigma_\pi(\vec{x}) := \sigma(\prod_{k=1}^{n} x_k) \ , \tag{2.4}$$

so lässt sich dies auch kurz schreiben als

$$H_\sigma := \overline{span} \left\{ \sigma_\pi(\rho(\vec{x} - \vec{d})) \ : \ \vec{d} \in \mathbf{R}^n, \ \rho \in \mathbf{R} \right\} = C(\mathbf{R}^n) \ , \tag{2.5}$$

wobei $\overline{span}$ den Abschluss bezüglich der Topologie der gleichmäßigen Konvergenz auf kompakten Mengen bezeichnet.

Beweis: Vergleiche die Originalarbeiten [Len94, Pin96] oder [Len03, S. 163ff]. ■

Auch Zeichnungen (vgl. Abbildung 2.1) können sinnvoll sein und eingebunden werden:

Abbildung 2.1: Reales Neuron (Schematische Skizze)

2.2 Die Topologie neuronaler Strukturen

2.3 Der Ausführ-Modus

2.4 Der Lern-Modus

Um Algorithmen zu erläutern kann es sinnvoll sein Pseudocode zu verwenden. Ein Beispiel dafür ist der Pseudocode 2.4.1.

Require: $n \geq 0 \vee x \neq 0$
Ensure: $y = x^n$

```
1:  y ⇐ 1
2:  if n < 0 then
3:      X ⇐ 1/x
4:      N ⇐ −n
5:  else
6:      X ⇐ x
7:      N ⇐ n
8:  end if
9:  while N ≠ 0 do
10:     if N ist gerade then
11:         X ⇐ X × X
12:         N ⇐ N/2
13:     else
14:         y ⇐ y × X
15:         N ⇐ N − 1
16:     end if
17: end while
```

Pseudocode 2.4.1: Berechne $y = x^n$

Kapitel 3

Implementierung neuronaler Netze

3.1 Die Wahl der Programmiersprache: Java

Kurze Begründung für die Auswahl der Programmiersprache. Mit welchem Entwicklungstool wurde gearbeitet und warum Oder wurde direkt mit dem JDK gearbeitet? Wenn ja, warum? **Keine** detaillierte Einführung in Java; das ist inzwischen Standard. Allerdings: Neue und spezielle Bibliotheken, Packages oder Klassen, die benutzt werden, müssen begründet und erläutert werden.

3.2 Details zum Entwurf und zur Entwicklung

Klassische Vorgehensweise bei dem Entwurf und der Entwicklung eines Anwendungsprogramms (OOA, OOD, OOP, etc.). Insbesondere sollten hier (oder – falls zu umfangreich – spätestens im Anhang) die entsprechenden Diagramme eingebunden werden.

3.3 Details zur konkreten Implementierung

Hier sollten ausgewählte Teile des Source-Codes, die für die Funktionalität des Programms fundamental sind, im Detail erläutert werden. Neben dem Aufzeigen der generellen Konzepte zur Umsetzung des mathematischen Kalküls in Programm-Code geht es hier auch um Fragen wie Effizienz, Parallelisierbarkeit, numerische Stabilität, etc..

```java
public final class HelloWorld
{
    /*
     * Ein Umlaut Test: Ä
     */
    public static void main(final String[] arg)
    {
        System.out.println("Hallo Welt!"); // Ausgabe: Hallo Welt!
    }
}
```

Quellcode 3.3.1: Ein Hallo-Welt-Programm in der Programmiersprache Java.

3.3.1 XML

Beispiel für XML-Code siehe Quelltext

```xml
<!-- Ein Kommentar in XML -->
<xs:element name="UsernameToken">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Username"/>
            <xs:element ref="Password" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="Id" type="xs:ID"/>
        <xs:anyAttribute namespace="##other"/>
    </xs:complexType>
</xs:element>
```

Quellcode 3.3.2: Beispiel für XML-Code

3.3.2 JAVA

Beispiel für Java-Code siehe Quelltext

```java
/**
 * JavaDoc
 */
public class JavaBeispiel implements garNichts {

/*
 * Das ist ein plumper Kommentar
 * der über zwei Zeilen geht
 */
    public void macheWas throws LatexException {
        for (int i = 0; i < 666; i++) {              //Schleife
    durchlaufen
            System.out.println("Mache was...");
        }
    }
}
```

Quellcode 3.3.3: Beispiel für Java-Code

3.4 Probleme bei der Implementierung

Klar!

3.5 Zeichnungen

Die folgenden Zeichnungen wurden mit den LaTeX-Zusatzpaketen pgf und tikz erstellt. Sie stellen sehr mächtige Werkzeuge zur Verfügung um Diagramme und Grafiken aller Art zu erstellen. Die Ergebnisse sind professionell und können, falls nötig, mit wenig Aufwand geändert werden. Es erfordert natürlich eine gewisse Einarbeitung, aber diese wird durch die Resultate schnell wieder aufgewogen. Eine umfangreiche Anleitung mit vielen weiteren Beispielen findet sich auf
http://www.ctan.org/tex-archive/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf

Es folgen einige Beispiele.

3.5.1 Zustandsdiagramm

Das Zustandsdiagramm (englisch: state diagram) der UML ist eine der dreizehn Diagrammarten dieser Modellierungssprache für Software und andere Systeme. Es stellt einen endlichen Automaten in einer UML-Sonderform grafisch dar und wird benutzt, um entweder das Verhalten eines Systems oder die zulässige Nutzung der Schnittstelle eines Systems zu spezifizieren.

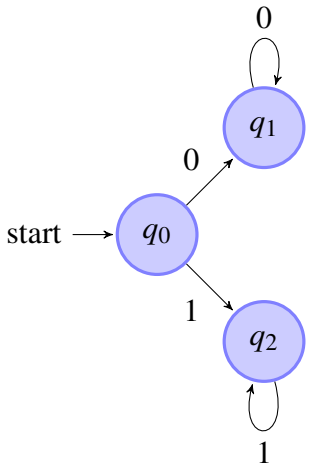

Abbildung 3.1: Zustandsdiagramm

3.5.2 Petrinetz

Ein Petri-Netz ist ein mathematisches Modell von nebenläufigen Systemen. Es ist eine formale Methode der Modellierung von Systemen bzw. Transformationsprozessen. Die ursprüngliche Form der Petri-Netze nennt man auch Bedingungs- oder Ereignisnetz. Petri-Netze wurden durch Carl Adam Petri in den 1960er Jahren definiert. Sie verallgemeinern wegen der Fähigkeit, nebenläufige Ereignisse darzustellen, die Automatentheorie.

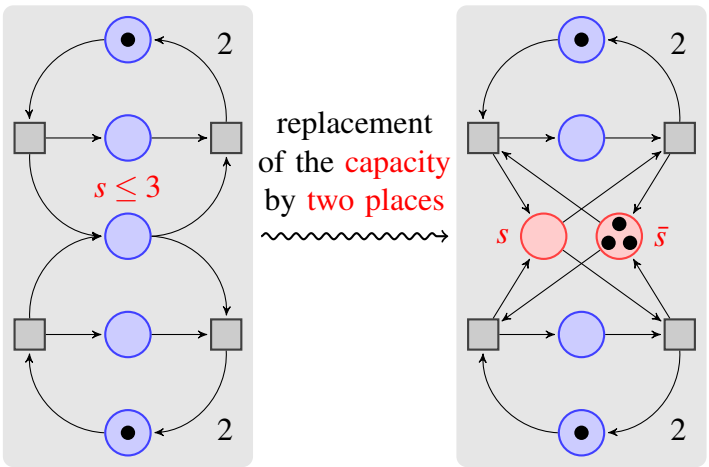

Abbildung 3.2: Petrinetz

3.5.3 Graph

Ein Graph besteht in der Graphentheorie anschaulich aus einer Menge von Punkten, zwischen denen Linien verlaufen. Die Punkte nennt man Knoten oder Ecken, die Linien nennt man meist Kanten, manchmal auch Bögen. Auf die Form der Knoten und Kanten kommt es im allgemeinen dabei nicht an. Knoten und Kanten können auch mit Namen versehen sein, dann spricht man von einem benannten Graphen.

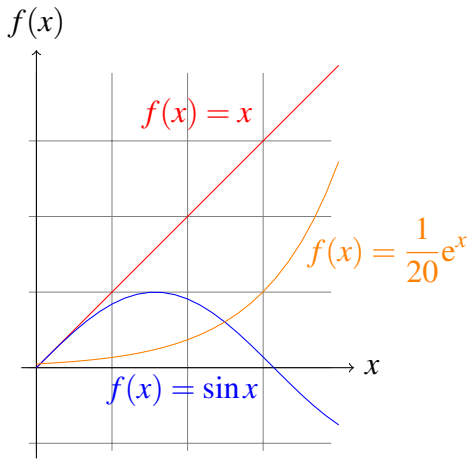

Abbildung 3.3: Graph

Kapitel 4

Anwendung neuronaler Netze

4.1 Anforderungen an Hard- und Software

- Hardware-Plattform
- Besonderheiten der Rechner-Architektur
- Prozessor-Typ bzw. -Typen
- Taktfrequenz
- Hauptspeicher-Größe
- Cache
- Swapspace (falls verwendet)
- Betriebssystem und Versionsnummer
- Entwicklungsumgebung
- Programmiersprache
- Compiler bzw. Interpreter und Versionsnummer
- Compiler- bzw. Interpreteroptionen
- verwendete Bibliotheken
- alle relevanten Implementationsdetails (insbesondere gesetzte Parameter u. ä.)
- usw., usw.

Alle diese Angaben sind wichtig, da sich die Ergebnisse sonst nur bedingt reproduzieren lassen. Nicht reproduzierbare Anwendungen sind aber wertlos!

4.2 Anwendung des Java-Programms

Hier sollte eine Beispiel-Anwendung mit dem entwickelten Programm durchgespielt werden und möglichst mit Screenshots dokumentiert werden.

Kapitel 5

Zusammenfassung und Ausblick

Noch einmal wird kurz erläutert, was in der Arbeit eigentlich gemacht wurde. Außerdem wird gesagt, was nach Meinung der Autorin bzw. des Autors noch alles gemacht werden könnte. Dadurch kann man zeigen, dass man auch etwas über den Tellerrand der eigentlichen Problemstellung hinweg gesehen hat. Insgesamt sollten dazu 1–2 Seiten genügen. Am Ende dieses Kapitels sollte die Bachelor-/Master-Arbeit etwa **40/80 Seiten** umfassen! Natürlich ist auch dies nur ein grober, aber im Auge zu behaltender Anhaltspunkt! Lieber gute 40 Seiten als redundante und langweilige 60 Seiten!

Literaturverzeichnis

[Len94] B. Lenze. Note on a density question for neural networks. *Numerical Funct. Analysis and Optimiz.*, 15:909–913, 1994.

[Len03] B. Lenze. *Einführung in die Mathematik neuronaler Netze*. Logos Verlag, Berlin, zweite auflage edition, 2003.

[Pin96] Allan Pinkus. Tdi-subspaces of $c(\mathbf{R}^d)$ and some density problems from neural networks. *Journal of Approximation Theory*, 85(3):269 – 287, 1996.

Anhang A

Diagramme und Tabelle

Ein oder mehrere Anhänge können, müssen aber nicht vorhanden sein. Als Faustregel gilt: Alles was den Lesefluss stört, kann in einen Anhang, also insbesondere Programmlistings (länger als eine Seite), umfangreiches Tabellen-Material, etc.. Die Listings der Programme, also der **Original-Source-Code**, sollten auf jeden Fall – außer eventuell im Anhang – auch auf CD-ROM der Bachelor-/Master-Arbeit in einem Einsteckfach beigelegt werden. Als Zugabe kann dort auch noch direkt ausführbarer Maschinen-Code für verschiedene Plattformen hinterlegt werden. Ebenfalls sollte es eine Selbstinstallationsroutine für mindestens ein gängiges Betriebssystem auf dem Datenträger geben!
Im Gegensatz zu normalen Kapiteln werden Anhänge zur besseren Unterscheidung nicht mit „1“, „2“, „3“,… durchnummeriert, sondern mit „A“, „B“, „C“,…. Ist nur ein Anhang vorhanden, kann die Nummerierung „A“ entfallen.

Am Ende des Anhangs sollte der Umfang der Bachelor-/Master-Arbeit etwa **60/100 Seiten** betragen!

Anhang B

UML-Diagramme

Hier könnten Klassen- oder UML-Diagramme stehen!

Anhang C

Quellcode

Hier könnten konkrete Teile des Java-Quellcodes stehen!