Dokumentation

Optimierung des Einkaufens

vorgelegt an der Technischen Hochschule Köln, Campus Gummersbach

im Fach Architektur verteilter Systeme

im Studiengang Informatik Schwerpunkt Software Engineering

vorgelegt von: Sven Thomas, 11117201

Philipp Felix Schmeier, 11117804

Sheila Kolodziej, 11143470

Ayhan Gezer, 11117870

Volkan Yüca, 11117315

Sümeyye Nur Öztürk, 11114188

Prüfer: Prof. Dr. Lutz Köhler (Technische Hochschule Köln)

Gummersbach, …

Technology
Arts Sciences
TH Köln

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Aufgabenstellung

Im Rahmen des Moduls „Architektur verteilter Systeme“ wird eine Anwendung zum optimierten Einkaufen erstellt.

Die Idee basiert auf einem alltäglichen Problem, das besonders in „fremden“ bzw. großen Einkaufsläden auftritt. Mit oder ohne Einkaufsliste ist es nahezu unmöglich den kürzesten Weg durch den Einkaufsladen zu finden und dabei an alle gewünschten Produkte zu gelangen. Meistens hat man zum Schluss immer noch ein paar Produkte offen und darf daraufhin im schlimmsten Fall noch einmal durch den gesamten Laden gehen, bevor man zur Kasse gehen darf. Dies ist ärgerlich und zeitaufwendig.

Die Anwendung soll dieses Problem beheben. Aus einer Auswahl von vordefinierten Einkaufsläden soll der Benutzer den gewünschten Laden auswählen können. Dort ist vermerkt, welches Produkt sich an welcher Stelle befindet. Der Benutzer muss daraufhin eine Einkaufsliste erstellen und anhand dieser beiden Komponenten wird der optimale Weg durch den Laden berechnet. Die Berechnung soll auf mehreren Rechnern stattfinden, um die Komplexität der verwendeten Algorithmen zeitsparend und qualitativ gewinnbringend zu verteilen. Anschließend wird dem Benutzer die Berechnung grafisch angezeigt, sodass dieser durch den Laden navigiert wird.

Aufgrund des eher geringen zeitlichen Rahmens dieser Veranstaltung wurde beschlossen, dass diese Aufgabe zunächst anhand des eher kleinen Lebensmittelladens „Dornseifer“ in Gummersbach bearbeitet wird.

2 Grundlegende Dokumente

2.1 Lastenheft

1. **Zielbestimmung**

Der Benutzer dieser Anwendung soll in der Lage sein mithilfe seiner eigenen Einkaufsliste den kürzesten Weg durch einen von ihm gewählten Lebensmittelladen zu finden.

2. **Produkteinsatz**

Das Produkt dient des zeitlich effizienten Einkaufs des Benutzers. Zielgruppe sind alle einkaufenden Kinder, Jugendliche und Erwachsene.

3. **Produktfunktionen**

/LF10/ Einkaufsliste erstellen /LF20/ Einkaufsliste verändern /LF30/ Lebensmittelladen auswählen /LF40/ Navigation durch den Lebensmittelladen

4. **Produktdaten**

/LD10/ Einkaufslistenelemente speichern /LD20/ Lebensmittelladenstruktur speichern

5. **Produktleistungen**

/LL10/ Flüssige und echtzeitnahe Navigation durch den Lebensmittelladen

6. **Qualitätsanforderungen**

Anforderung	**sehr gut**	**gut**	**normal**
Performance	x		
Erweiterbarkeit	x		

Tabelle 1: Qualitätsanforderungen

2.2 Pflichtenheft

1. **Zielbestimmung**

Der Benutzer dieser Anwendung soll in der Lage sein mithilfe seiner eigenen Einkaufsliste den kürzesten Weg durch einen von ihm gewählten Lebensmittelladen zu finden.

(a) **Musskriterien**

i. Benutzer kann Einkaufsliste erstellen
ii. Benutzer kann Einkaufsliste speichern
iii. Benutzer kann Einkaufsliste vor dem Start der Navigation verändern
iv. Lebensmittelladenstruktur von „Dornseifer“ in Anwendung enthalten
v. Lebensmittelladenstruktur von „Dornseifer“ wird in 2D bzw. 3D dargestellt
vi. Benutzer kann „Dornseifer“ als Lebensmittelladen auswählen
vii. Der zeitlich effizienteste Weg durch den Lebensmittelladen wird ermittelt
viii. Grafische Darstellung der Navigation durch den Lebensmittelladen
ix. Möglichkeit zur (zukünftigen) Erweiterung der Auswahl von Lebensmittelläden

(b) **Wunschkriterien**

i. Während der Navigation kann der Benutzer die Einkaufsliste anpassen
ii. Weitere Lebensmittelläden stehen zur Auswahl
iii. Anzeigen der zeitlichen Dauer des Einkaufs
iv. Integration in eine App

2. **Produkteinsatz**

(a) **Anwendungsbereiche**

i. Einkauf
ii. Alltag

(b) **Zielgruppe**

i. Alle einkauffähigen Kinder, Jugendliche und Erwachsene

(c) **Betriebsbedingung**

i. Alltägliche Umgebung

3. **Produktfunktionen**

/F10/ (/LF10/)

Einkaufsliste erstellen: Von Programmstart bis Navigationsstart

Ziel: Erstellen einer neuen Einkaufsliste

Vorbedingung: Navigation wurde noch nicht gestartet.

Nachbedingung Erfolg: Eine Einkaufsliste wurde erstellt.

Nachbedingung Fehlschlag: Eine Einkaufsliste konnte nicht erstellt werden.

Akteure: Benutzer

Auslösendes Ereignis: Benutzer drückt Button “Neue Einkaufsliste”

Beschreibung:

(a) Neue/Leere Einkaufsliste anzeigen

(b) Einkaufslistenelemente hineinschreiben lassen

(c) Einkaufsliste wird gespeichert

/F20/ (/LF20/)

Einkaufsliste verändern: Von Programmstart bis Navigationsstart

Ziel: Benutzer verändert eine vorhandene Einkaufsliste

Vorbedingung: Eine Einkaufsliste ist vorhanden

Nachbedingung Erfolg: Einkaufsliste wurde verändert

Nachbedingung Fehlschlag: Einkaufsliste konnte nicht verändert werden

Akteure: Benutzer

Auslösendes Ereignis: Benutzer drückt Button „Einkaufsliste bearbeiten“

Beschreibung:

(a) Benutzer wählt eine Einkaufsliste aus

(b) Ausgewählte Einkaufsliste wird angezeigt

(c) Benutzer verändert Einkaufsliste

(d) Veränderungen werden gespeichert

F30/ (/LF30/)

Lebensmittelladen auswählen: Von Programmstart bis Navigationsstart

Ziel: Benutzer wählt einen Lebensmittelladen aus

Vorbedingung: Mindestens ein Lebensmittelladen wurde in die Anwendung integriert

Nachbedingung Erfolg: Ein Lebensmittelladen wurde ausgewählt

Nachbedingung Fehlschlag: Es konnte kein Lebensmittelladen ausgewählt werden

Akteure: Benutzer

Auslösendes Ereignis: Benutzer drückt Button „Lebensmittelladen auswählen“
Beschreibung:

(a) Benutzer wählt einen vordefinierten Lebensmittelladen aus
(b) Auswahl wird gespeichert

F40/ (/LF40/)
Navigation durch den Lebensmittelladen: Von Navigationsstart bis Navigationssende
Ziel: Der zeitlich effizienteste Weg durch den Lebensmittelladen wird berechnet und dem Benutzer angezeigt
Vorbedingung: Einkaufsliste und Lebensmittelladen wurden ausgewählt
Nachbedingung Erfolg: Der Benutzer wird durch den Lebensmittelladen navigiert
Nachbedingung Fehlschlag: Die Navigation kann nicht gestartet werden
Auslösendes Ereignis: Spieler drückt Button „Navigation starten“
Beschreibung:

(a) Anhand der Einkaufslistenelemente wird der zeitlich effizienteste Weg von Ladeneingang bis zur Ladenkasse ermittelt.
(b) Die Einkaufslistenelemente werden umsortiert und in eine Abholreihenfolge eingereiht
(c) Der Benutzer erhält eine grafische Navigation zu dem ersten Abholelement mittels einer Linie und einem Bild des Einkaufslistenelements
(d) Benutzer folgt Navigation
(e) Benutzer vermerkt Erfolg durch Anklicken des „Häkchen“-Symbols oder Misserfolg durch Anklicken des „Kreuz“-Symbols
(f) Erfolg/Misserfolg wird in der Einkaufsliste gespeichert durch „Häkchen“ oder „Kreuz“ Symbol hinter dem jeweiligen Einkaufslistenelement
(g) Der Benutzer erhält eine grafische Navigation zu dem nächsten Abholelement mittels einer Linie und einem Bild des Einkaufslistenelements
(h) Wurde das letzte Abholelement erfolgreich/nicht erfolgreich in den Einkaufswagen gelegt, wird die Navigation beendet.

4. **Produktdaten**

/D10/ (/LD10/)
Name des jeweiligen Lebensmittels aus der Einkaufsliste

/D20/ (/LD20/)

Lebensmittelladenstruktur:

(a) Position der Regale, Wände, Kassen, Ein-/Ausgang

(b) Position und Name der Lebensmittel

5. **Produktleistungen**

/L10/ (/LL10/)

Der aktuelle Standort des Benutzers muss regelmäßig/echtzeitnah aktualisiert werden

6. **Qualitätsanforderungen**

Anforderung	**sehr gut**	**gut**	**normal**
Performance	x		
Erweiterbarkeit	x		

Tabelle 2: Qualitätsanforderungen

7. **Benutzeroberfläche**

(a) Design ist dem Entwickler überlassen

(b) Benutzerinteraktion ist dem Entwickler überlassen

8. **Technische Produktumgebung**

(a) Android

2.3 Glossar

Master-Worker-Architektur

Diese Architektur zum hierarchischen Verwalten des Zugriffs auf eine gemeinsame Ressource besitzt viele unterschiedliche Namen. Im Rahmen dieser Veranstaltung wird nur der Begriff der Master-Worker-Architektur verwendet.

Einkaufslistenelement

Ein Element, das vom Benutzer in einer Einkaufsliste eingetragen worden ist. Es besteht jeweils nur aus einem Produkt z.B. Salami.

Abholelement

Ein Element, das vom Benutzer in einer Einkaufsliste als Einkaufslistenelement eingetragen worden ist und einen Einkaufsknoten im Navigationsgraph darstellt. Es wird vom Benutzer im Laufe der Navigation/des Einkaufens eingekauft.

Einkaufsknoten

Jedes einzelne Einkaufslistenelement beschreibt einen Einkaufsknoten bei der Berechnung und Navigation der optimierten Strecke durch den Lebensmittelladen. Im Optimalfall besucht der Benutzer jeden einzelnen Knoten und kauft bei n Einkaufsknoten n verschiedene Produkte ein.

(Berechnungsknoten)

Unterstützende Knoten bei der Berechnung der optimalen Strecke. Sie helfen dabei Abbiegungen oder schwierige Wege kollisionsfrei zu berechnen.

Navigationsgraph

Der Navigationsgraph beschreibt die optimale Strecke vom Ladenanfang bis zum Ladenende und besitzt n Knoten für n Produkte, die auf der Einkaufsliste stehen.

2.4 Code Convention

Da die Komponenten in verschiedenen Sprachen geschrieben worden sind, kann man sich nicht auf einen einheitlichen Standard festsetzen. Für die jeweilige Programmiersprache werden die offiziellen Code Conventions als Standard angesehen und für die jeweilige Komponente eingehalten.
Kommentare sollen auf Englisch verfasst werden. Nach Möglichkeit sollte jede Methode mit einer Dokumentation versehen werden. Dazu gehören eine kurze Beschreibung der Aufgabe der Methode, sowie ggf. eine Erläuterung für die jeweiligen Parameter und den Rückgabewert der Funktion. Bei *trivialen* Methoden (z.B. Getter/Setter) kann darauf verzichtet werden.

3 Recherche und Grundentscheidung

3.1 Algorithmen

3.1.1 Travelling Salesman Problem (TSP)

Das Problem des Handlungsreisenden auch genannt als *Travelling Salesman Problem* ist ein kombinatorisches Optimierungsproblem unter Anderem in der theoretischen Informatik. Der „Handlungsreisende“ möchte verschiedene Orte besuchen. Die Aufgabe des Problems besteht darin, eine Reihenfolge mehrerer Orte so zu wählen, dass kein Ort außer dem ersten mehr als einmal besucht wird. Außerdem soll dadurch die Strecke des Handlungsreisenden so kurz wie möglich sein und einen Kreis bilden, d.h. der erste Ort ist gleich dem letzten Ort [WI01].[1]

So wie bei vielen mathematischen Lösungen wird auch hier diese *reale* Situation durch ein einfacheres Modell abgebildet. Das Problem des Handlungsreisenden lässt sich mit einem Graphen modellieren. Die Knoten repräsentieren die Orte zu denen der Handlungsreisende möchte, während die Kanten eine Verbindung zwischen diesen Städten beschreibt. Zur Vereinfachung kann der Graph als **vollständig** angenommen werden, d.h. zwischen zwei Knoten existiert immer eine Kante. Unter Umständen muss dann eine künstliche Kante eingefügt werden. Diese wird bei der Berechnung der kürzesten Strecke nicht mitberücksichtigt, da die Länge einer solchen Querverbindung niemals zu einer kürzesten Strecke beitragen würde.

Man unterscheidet zwischen asymmetrischem und symmetrischem Travelling Salesman Problem. Beim asymmetrischen können die Kanten in Hin- und Rückrichtung unterschiedliche Längen besitzen. Deswegen wird diese Situation dann mit einem gerichteten Graphen modelliert, um die Richtung ebenfalls angeben zu können.

Beim symmetrischen Travelling Salesman Problem die Kantenlängen in beiden Richtungen identisch. Deswegen wird meist ein ungerichteter Graph zur Modellierung verwendet[WI01][2]

Bei *n* Städten beträgt die Anzahl der möglichen Wege *n!*, d.h. bei sehr hohen *n* wird es unmöglich in einer angenehmen Zeit alle Wege zu berechnen. Hierbei kann paralleles Rechnen die gesamte Rechenzeit reduzieren[BO06][3].

[1] [WI01] Wikipedia: Problem des Handlungsreisenden https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden (Stand: 12.11.2019).

[2] Vgl. Ders. (Stand: 12.11.2019)

[3] [BO06] Borovska, Plamenka: Solving the Travelling Salesman Problem in Parallel by Genetic Algorithm on Multicomputer Cluster; International Conference on Computer Systems and Technologies, 2006.

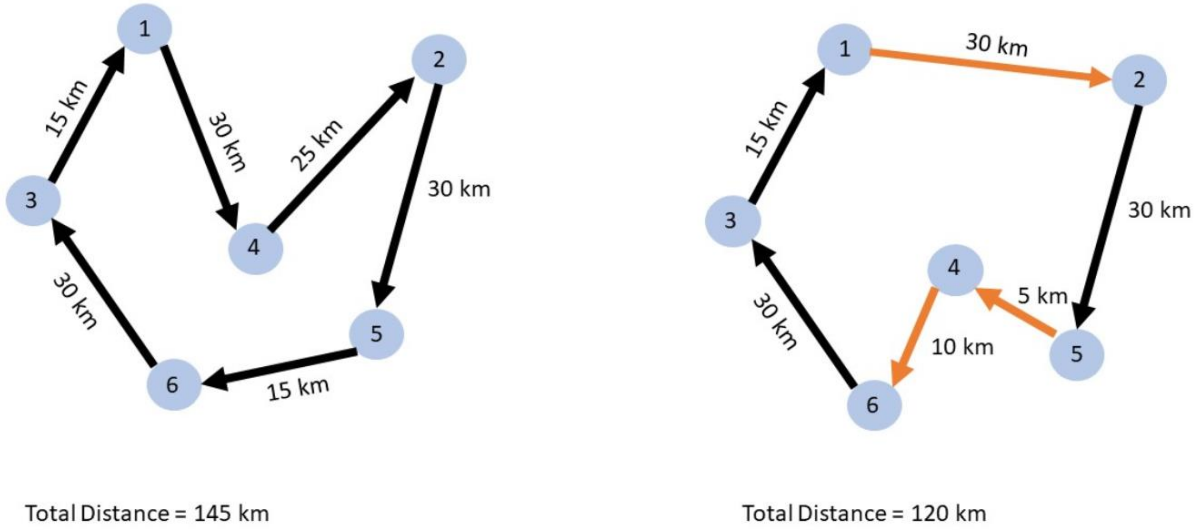

Abbildung 1: Travelling Salesman Problem

3.1.2 Genetische Algorithmen (GA)

Genetische Algorithmen gehören zu den sogenannten evolutionären Algorithmen. Ihre Idee ist es evolutionäre Prozesse nachzuahmen. Der Aufbau eines genetischen Algorithmus sieht wie folgt aus: Zunächst wird das zu optimierende Problem kodiert, d.h. auf ein binär kodiertes Chromosom abgebildet. Dann initialisiert man eine Ausgangspopulation, indem eine Population von Individuen erzeugt und zufällig initialisiert wird. Man spricht hier von der Generation 0. Jedes Individuum wird daraufhin mit einer Fitnessfunktion bewertet, d.h. jedes einzelnem Chromosom wird eine reellwertige Zahl zugeordnet.
Daraufhin werden jeweils zwei Elternteile mittels einer Selektionsvariable selektiert und es entstehen mittels einer gewählten Kreuzungsvariante Nachkommen dieser Elternteile. Da die Allele der Nachkommen mutieren können, werden die Werte invertiert.
Dann wird die Population um den neuen Nachkommen ergänzt. Wird dadurch die Populationsgröße überschritten werden mittels Ersetzungsstrategien alte Nachkommen durch neue ersetzt.
Diese Verfahrensschritte werden solange wiederholt bis ein Abbruchkriterium erfüllt ist. Das Abbruchkriterium sollte geschickt gewählt werden. Allein das Erreichen der Zielfunktion wird in den meisten Fällen nicht reichen, da genetische Algorithmen stochastische Suchalgorithmen sind, diese also das absolute Optimum nie erreichen würden. Das Kriterium sollte demnach einen Abbruch bei hinreichender Nähe zur Zielfunktion zulassen Um ein zeitliches Abbruchkriterium zu erhalten kann man außerdem festlegen, dass man nach einer

maximalen Anzahl von Generationen aufhört. Dies bietet sich an, wenn der genetische Algorithmus sich dem Optimum nicht schnell genug nähert [KE04][4].

3.1.3 Kombination aus TSP und GA

Bei einer Kombination aus dem Travelling Salesman Problem und genetischem Algorithmus bestehen die Individuen aus einem String aus Zahlen. Jede Zahl repräsentiert die Orte die der Handelsreisende besuchen möchte. Außerdem repräsentieren die Zahlen die jeweilige Position.
Die Fitness Funktion repräsentiert die Länge der Reise. Dabei kann z.B. die euklidische Distanz zur Berechnung eines jeden Weges herangezogen werden. Die Auswahl erfolgt meistens über eine zufällige Wahl von *k* Individuen aus denen die zwei fittesten zur Kreuzung ausgewählt werden. Die Kreuzung basiert über ein zufälliges Crossover-Prinzip, d.h. zwei Teile der Eltern werden mittels des Crossover-Punktes kopiert und es entstehen zwei Kinder. Die andere Hälfte des jeweiligen Kindes wird mit den fehlenden Städten des zweiten Elternteils aufgefüllt (siehe Abb. 3).
Die mögliche Mutation wird durch ein zufälliges Ändern der Ortreihenfolge hervorgerufen.

3.2 Architektur

3.2.1 Mögliche Architekturen bei verteilten Systemen und ihre Anwendung

(Quelle...) In verteilten Systemen kommen mittlerweile viele verschiedene Architekturen zum Einsatz. Diese werden im folgenden Abschnitt kurz erläutert. Hierbei werden nicht alle möglichen Architekturen beschrieben, sondern nur die, die im Rahmen dieser Veranstaltung sinnvoll genutzt werden konnten.

[4] [KE04] Keen, Shawn: Ausarbeitung zu Genetischen Algorithmen http://www.informatik.uni-ulm.de/ni/Lehre/SS04/ProsemSC/ausarbeitungen/Keen. pdf (Stand:13.11.2019).

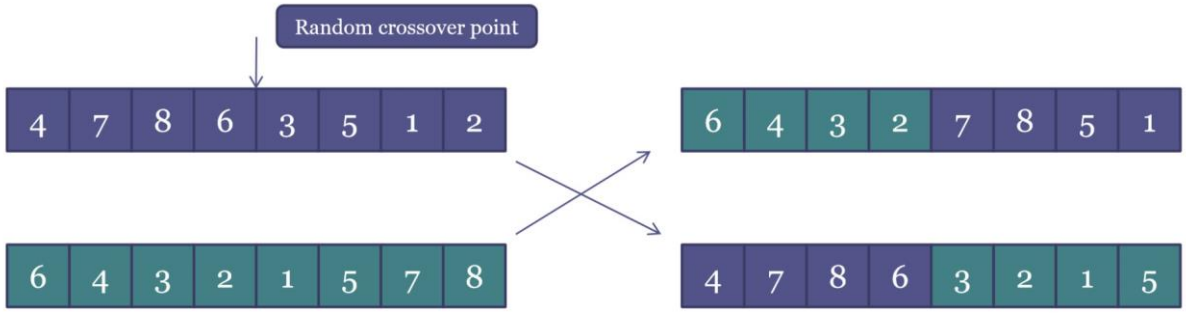

Abbildung 2: Kreuzung zweier Eltern

Die wohl bekannteste Architektur ist die **Master-Slave-/Master-Worker-Architektur**. Wie der Name vermuten lässt gibt es einen Teilnehmer mit der Rolle des Masters und viele Teilnehmer mit der Rolle des Workers. Nur der Master kann auf unaufgefordert eine gemeinsame Ressource zugreifen. Die Worker können dies erst, wenn sie vom Master dazu aufgefordert werden (Polling). Vorteil ist hier, dass die Zugriffsverhältnisse klar aufgeteilt sind. Allerdings ist diese Architektur ein wenig einschränkend, da die Worker nicht untereinander kommunizieren können und das Polling der Worker durch den Master ineffizient ist.
Eine etwas gleichberechtigtere Variante vom Master-Worker -Prinzip ist die **Peer-toPeer** Verbindung, wo alle Teilnehmer gleichberechtigt sind und sowohl Dienste in Anspruch nehmen als auch Dienste zur Verfügung stellen können. In der Praxis werden aber auch hier die Rechner oft in Aufgabenbereiche eingeteilt um Ordnung in die Aufgabenverteilung zu schaffen.
Eine dritte Möglichkeit ist die **N-Tier-Architektur**. Dies ist eine Schichtenarchitektur, in der Aspekte des Systems einer Schicht zugeordnet werden. Bei einer drei-SchichtenArchitektur hat man beispielsweise eine Präsentations-, Anwendungs- und Persistenzschicht. Diese Art der Architektur eignet sich vor allem für komplexe Softwaresysteme. Vorteil ist hier eine zentrale Datenhaltung, sowie die Skalierbarkeit, da man beliebig viele Schichten konstruieren kann. Aber auch die Fehlertoleranz ist hier als deutlich verbessert als in anderen Architekturen.
Eine weitere Möglichkeit ist eine **Serviceorientierte(SOA)-Architektur**. Hier wird sich oft an Geschäftsprozessen orientiert, deren Abstraktion die Grundlage für Serviceimplementierung liefert. Maßgeblich sind hier nicht die technischen Einzelaufgaben, sondern die Zusammenfassung dieser Aufgaben zu einer größeren, komplexeren Aufgabe. Dadurch wird die Komplexität der einzelnen Anwendungen hinter den standardisierten Schnittstellen

verborgen. Vorteile dieser Architektur sind eine hohe Flexibilität und eine mögliche Senkung der Programmierkosten, da bestimmte Services ggf. wiederverwendet werden können.
Um das Zusammenspiel der einzelnen Komponenten durch Ereignisse zu steuern, wurde die **Event-driven-Architektur** entwickelt. Dabei kann ein Ereignis sowohl von außen als auch vom System selbst ausgelöst werden. Voraussetzung zum Einsatz dieser Architektur ist allerdings, dass alle Teilnehmer, die bei der Abwicklung des Ereignisses beteiligt sind, miteinander kommunizieren können. Sie kann als Ergänzung zur service-orientierten Architektur verwendet werden, da auch dort Dienste durch Ereignisse ausgelöst werden können.

3.2.2 Wahl der Architektur

Hinsichtlich der zuvor vorgestellten Architekturen und der zuvor beschriebenen Aufgabenstellung kann man die N-Tier-Architektur nicht empfehlen. In diesem Projekt ist eine hohe Skalierbarkeit nicht notwendig. Ebenso bietet eine serviceorientierte Architektur für dieses Projekt keinen Mehrwert, da sich die Flexibilität unserer Anwendung lediglich auf die unterschiedlichen Einkaufsläden beschränkt. Als Ergänzung zur serviceorientierten Architektur entfällt damit auch die Event-driven-Architektur. Sie könnte zwar durchaus abgrenzend zur SOA genutzt werden, aber hinsichtlich der verwendeten Algorithmen bietet sich eher eine Master-Worker oder eine Peer-to-Peer Architektur an.

Betrachtet man den genetischen Algorithmus genauer, stellt man fest, dass es reicht, wenn die Kommunikation nur zwischen dem Master und den Workern stattfindet. Die Worker berechnen die neue Generation und müssen das Ergebnis nur dem Master mitteilen. Eine Kommunikation zwischen den einzelnen Workern ist demnach nicht notwendig, somit wird in diesem Projekt die Master-Worker Architektur angewendet.

Außerdem ist diese Architektur so bekannt und verbreitet, dass eine Implementierung in den meisten Programmiersprachen keinen großen Aufwand darstellt.

4 Entwurfsdiagramme

4.1 Architekturdiagramm

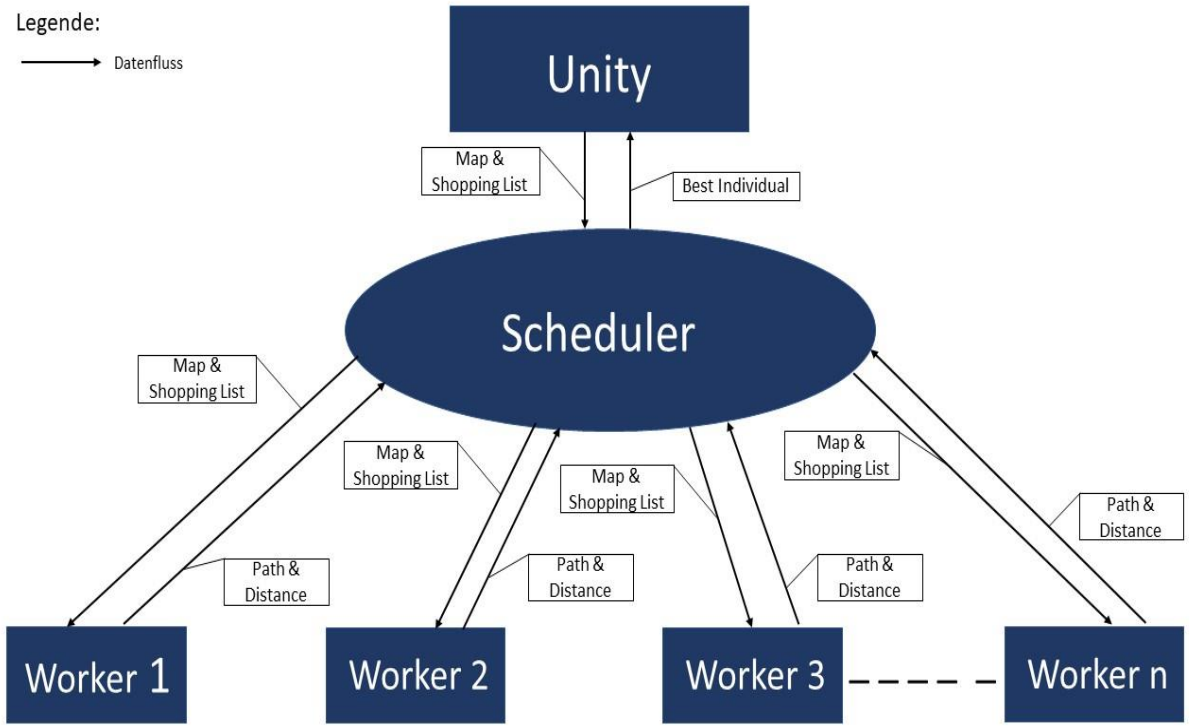

Abbildung 3: Architekturdiagramm

Abbildung 1 beschreibt die Architektur der Anwendung. In Unity wird die Karte/Plan von „Dornseifer“ erstellt und die Möglichkeit angeboten, eine Einkaufsliste zu erstellen. Nachdem man die Einkaufsliste erstellt hat, wird diese Einkaufsliste zusammen mit der Karte an den Scheduler geschickt. Der Scheduler dient als Schnittstelle zwischen der Unity - Instanz und den Workern. Dieser übergibt die Karte und die Einkaufsliste an die einzelnen Worker, die diese Individuen abfragen. Die Worker berechnn (mittels Genetische Algorithmen und Travelling Salesman Problem) den Pfad und die kürzeste Distanz. Diese werden anschließend an den Scheduler weitergegeben, der die kürzeste Distanz von allen Workern bestimmt und an Unity übergibt, welche das Ergebnis visualisiert.

5 Komponentenaufbau

5.1 Grafische Oberfläche

Abbildung 4: Unity-Logo

Für die graphische Oberfläche wurde die Anwendung Unity verwendet. Mit Unity können neben eigene Spiele auch 3D-Modelle, Texturen, Animationen und viele andere komplexe Projekte realisieren.
Das Frontend bildet eine Unity-Instanz. In dieser sind die verschiedenen Läden mit den Artikelstandorten in einer 3D-Welt abgebildet. Ebenso wird hier die Einkaufsliste erstellt und das Ergebnis präsentiert. Somit stellt die Unity Anwendung sowohl den Start- als auch den Endpunkt der verteilten Anwendung dar.

Die Funktionalitäten, die das Unity Frontend anbietet sind:

(1) Visuelle Abbildung der Läden mit Artikelstandorten.
(2) Generierung einer abstrahierten Darstellung eines Ladens für den Algorithmus.
(3) Bereitstellung eines UI zur Erstellung einer Einkaufsliste.
(4) Start des Algorithmus durch Übergabe der Daten an den Scheduler.
(5) Visualisierung des Ergebnisses.

5.1.1 Abbildung der Läden mit Artikelstandorten

Um die Frontend in Unity umzusetzen wurde zuerst eine Skizze von dem Einkaufsladen „Dornseifer“ erstellt. Die einzelnen Artikeln wurden zuerst in Kategorien zusammengefasst. Bei Kategorien, die mehrmals auftreten, wird später eine Spezifikation erfolgen.

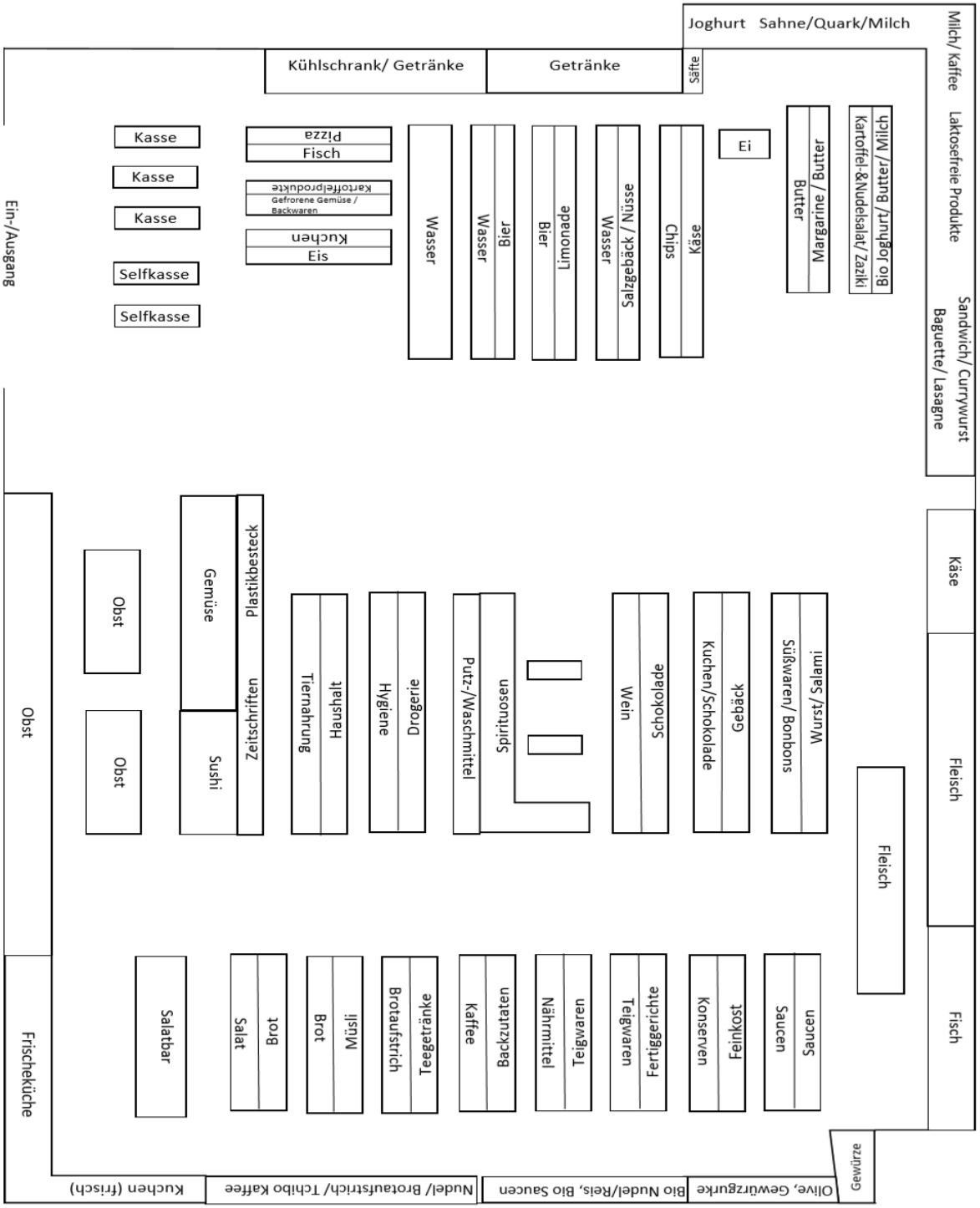

Abbildung 5: Skizze von Dornseifer

In Abbildung 5 sieht man die obige Skizze von dem Einkaufsladen in Unity umgesetzt. Die einzelnen Produktkategorien besitzen unterschiedliche Farben, welche in der Legende aufgelistet sind. Bei der Berechnung des optimalen Weges wird als Startpunkt immer der Eingang und als Endpunkt die Kassen verwendet. Zwischen den Regalen wurden jeweils Punkte gesetzt, anhand denen man die Regale durchlaufen kann. Drückt man oben rechts auf den Button „Einkauf planen“, öffnet sich der Einkaufsplaner.

Abbildung 6: Übersicht Dornseifer

Ist der Einkaufsplaner nun geöffnet, kann man die einzelnen Artikel, die man sich wünscht durch anklicken in die Einkaufsliste einfügen. Wenn man nun auf den „Starte Berechnung“ Button drückt, wird die Karte und die Einkaufsliste an den Scheduler weitergegeben, die diese an die Worker weiterleitet.

Abbildung 7: Einkaufsplaner

Nachdem man den optimalen Weg von dem Scheduler zurückbekommt, wird die Einkaufsliste sortiert ausgegeben. Um den optimalen Weg zu erhalten, sollte der Benutzer die Einkaufsliste in dieser Reihenfolge abarbeiten.

Abbildung 8: Ergebnis optimaler Weg

Drückt man nun auf den Button „Visualisiere“, erhält man eine Visualisierung des Ergebnisses. Dabei geht die orangene Figur durch den Einkaufsladen und zeigt den optimalen Weg.

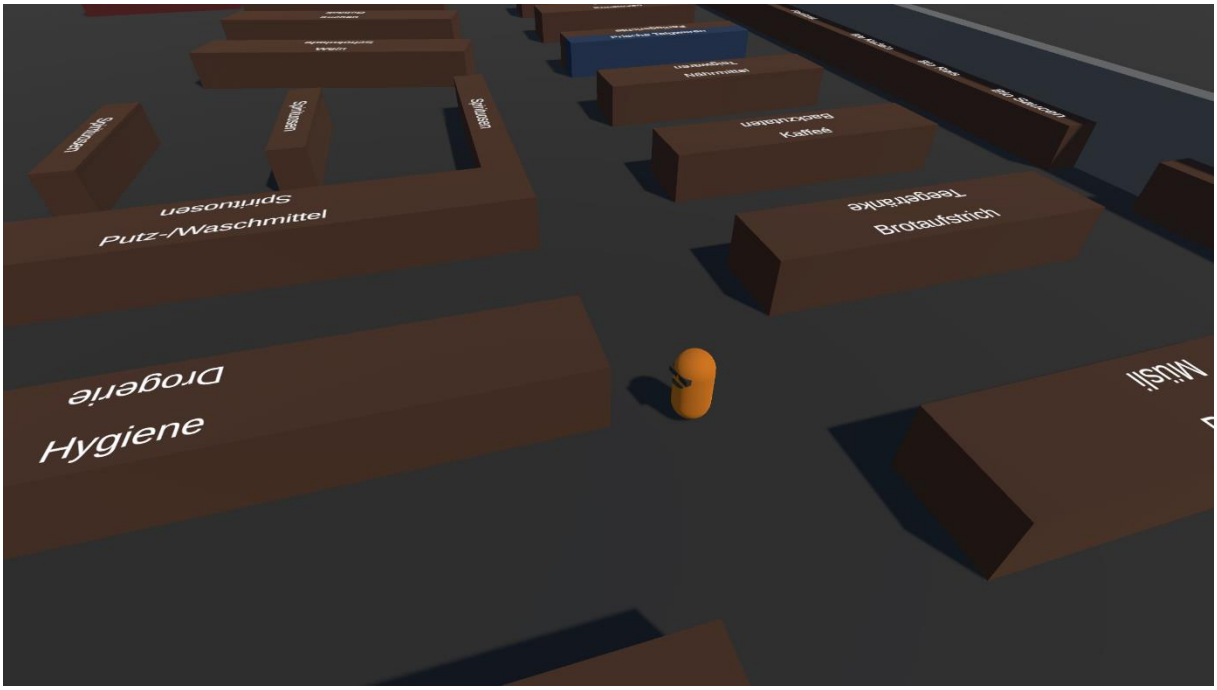

Abbildung 9: Visulisierung des Ergebnisses

5.2 Scheduler

Der Scheduler bildet die Schnittstelle zwischen der Unity-Instanz und den einzelnen Workern ab. Über eine REST-API (siehe Scheduler Ressourcentabelle) können sich unter anderem die Worker anmelden und die Daten aktualisieren.

Die Scheduler Ressourcentabelle beschreibt, auf welche Daten die einzelnen Komponenten Zugriff haben und beschreibt deren Beziehung.

Ressource	Verb	URI	Semantik	Contenttype-Request	Contenttype-Response
Worker					
	Post	/worker	Erstellt einen Worker und gibt UUID und Population zurück	application/json	application/json
	Put	/worker?uuid={parameter}	Akutalisiert den entsprechenden Worker und liefert eine neue Population zurück	applicaiton/json	applicaiton/json
Map					
	Get	/map	Liefert die Karte der Unity Instanz		application/json
	Post	/map	Setzt die Karte der Unity Instanz	application/json	
Path					
	Get	/path	Liefert den aktuell besten Weg zurück.		application/json

Tabelle 3: Scheduler Ressourcentabelle

Die HTTP-Fehlercodes werden auf HTTP-Anfragen von einem Server als Antwort zurückgegeben, ob es zu einem Fehler beim Aufruf gekommen ist. Der Fehlercode gibt Informationen darüber, welcher Fehler bei der Ausführung aufgetreten ist.

Ressource	Verb	Fehlercode		Beschreibung
/worker	POST	503	Service Unavailable	Die Map ist nicht gesetzt.
				Die maximale Anzahl der Arbeiter wurde erreicht.
	PUT	400	Bad Request	Population ist nicht valide (enthält z.B. null-Werte).
				Das JSON konnte nicht gelesen werden.
				Die UUID fehlt.
				Der Nutzer ist nicht registriert.
/map	GET	204	No Content	Die Map ist nicht verfügbar.
	POST	400	Bad Request	Das JSON konnte nicht gelesen werden.
/path	GET	503	Service Unavailable	Es gibt momentan keine registrieren Worker, die an der Lösung arbeiten.
		204	No Content	Die Worker haben noch kein Ergebnis geliefert.

Tabelle 4: Scheduler HTTP-Fehlercodes

5.3 Master-Worker

Worker steht ständig in Verbindung mit dem Scheduler, um neue Individuen abfragen zu können. Jedes Worker berechnet mittels Genetische Algorithmen und Travelling Salesman Problem den besten Weg und liefert diese an den Scheduler. Zurzeit werden 100.000 Generationen verwendet. Dies dauert in der Regel 5 Sekunden.

6 Quellenverzeichnis

6.1 Literatur

[BO06] Borovska, Plamenka: Solving the Travelling Salesman Problem in Parallel by Genetic Algorithm on Multicomputer Cluster; International Conference on Computer Systems and Technologies, 2006.

6.2 Internetquellen

[WI01] Wikipedia: Problem des Handlungsreisenden https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden (Stand: 12.11.2019).

[KE04] Keen, Shawn: Ausarbeitung zu Genetischen Algorithmen http://www.informatik.uni-ulm.de/ni/Lehre/SS04/ProsemSC/ausarbeitungen/Keen. pdf (Stand: 13.11.2019).

A Anhang

A.1 Ergebnisprotokolle

A.2 Zeitprotokolle