****

**FH - Studiengang für**

**Informationstechnik und System-Management**

**Salzburg**

**ITS**

**Übungen in**

**Spezielle Softwaretechnologien**

**Protokoll**

Gegenstand der Übung gemäß Anleitung:

**Verkehrssimulation**

**Version: 1**

**Datum der Übung: 01.03.2017**

**Datum der Abgabe: 13.07.2017**

**Autoren: Christopher Wieland, Martin Wieser, Stephanie Kaschnitz, Hannes Kleiner, Andreas Lippmann**

Inhaltsverzeichnis

[1 Aufgabenstellung 1](#_Toc487555025)

[2 Gesamtübersicht und Idee 2](#_Toc487555026)

[2.1 GUI 2](#_Toc487555027)

[2.1.1 Klasse „Verkehrsteilnehmer“ 2](#_Toc487555028)

[2.1.2 Klasse „Ampel“ 2](#_Toc487555029)

[2.1.3 Klasse „Obstacle“ 2](#_Toc487555030)

[2.2 Verkehrsnetz 3](#_Toc487555031)

[2.2.1 Klasse „1“ 3](#_Toc487555032)

[2.2.2 Klasse „2“ 3](#_Toc487555033)

[2.3 Verkehrsregeln 3](#_Toc487555034)

[2.3.1 Klasse „1“ 3](#_Toc487555035)

[2.3.2 Klasse „2“ 3](#_Toc487555036)

[2.3.3 Klasse „2“ 3](#_Toc487555037)

[2.4 Verkehrsteilnehmer 3](#_Toc487555038)

[2.4.1 Klasse „1“ 3](#_Toc487555039)

[2.4.2 Klasse „2“ 4](#_Toc487555040)

[2.4.3 Klasse „2“ 4](#_Toc487555041)

[2.5 Ampelsteuerung 4](#_Toc487555042)

[2.5.1 Klasse „Ampeln“ 4](#_Toc487555043)

[2.5.2 Klasse „Ampelsteuerung“ 4](#_Toc487555044)

[2.6 RabbitMQ 4](#_Toc487555045)

[2.6.1 Klasse „1“ 4](#_Toc487555046)

[2.6.2 Klasse „2“ 5](#_Toc487555047)

[2.6.3 Klasse „2“ 5](#_Toc487555048)

[3 Dokumentation der Funktionalität der Verkehrssimulation 6](#_Toc487555049)

[3.1 GUI 6](#_Toc487555050)

[3.1.1 Funktionen der Klasse Verkehrsteilnehmer 6](#_Toc487555051)

[3.1.2 Funktionen der Klasse ObjectHandler 7](#_Toc487555052)

[3.1.3 Funktionen der Klasse Ampel 7](#_Toc487555053)

[3.1.4 Funktionen der Klasse AmpelHandler 9](#_Toc487555054)

[3.1.5 Funktionen der Klasse Obstacle 9](#_Toc487555055)

[3.1.6 Funktionen der Interfaces IObject 10](#_Toc487555056)

[3.1.7 Funktionen der Interfaces IVerkehrsnetz 10](#_Toc487555057)

[3.2 Verkehrsnetz 11](#_Toc487555058)

[3.2.1 Funktionen 11](#_Toc487555059)

[3.2.2 Deklarierung der Funktionen 12](#_Toc487555060)

[3.3 Verkehrsregeln 12](#_Toc487555061)

[3.3.1 Funktionen 12](#_Toc487555062)

[3.3.2 Deklarierung der Funktionen 13](#_Toc487555063)

[3.4 Verkehrsteilnehmer 13](#_Toc487555064)

[3.4.1 Funktionen 13](#_Toc487555065)

[3.4.2 Deklarierung der Funktionen 14](#_Toc487555066)

[3.5 Ampelsteuerung 14](#_Toc487555067)

[3.5.1 Funktionen der Klasse Ampel 14](#_Toc487555068)

[3.5.2 Funktionen der Klasse Ampelsteuerung 15](#_Toc487555069)

[3.5.3 Variablen der Klasse Ampelsteuerung 17](#_Toc487555070)

[3.6 RabbitMQ 17](#_Toc487555071)

[3.6.1 Funktionen 17](#_Toc487555072)

[3.6.2 Deklarierung der Funktionen 18](#_Toc487555073)

[4 Zusätzliche externe Komponenten 18](#_Toc487555074)

[5 Zusammenfassung und Ausblick 19](#_Toc487555075)

[5.1 Derzeit nicht implementiert 19](#_Toc487555076)

[5.2 Ausblick 19](#_Toc487555077)

Tabellenverzeichnis

[Tabelle 3.1: Funktionen der GUI 8](#_Toc487555078)

[Tabelle 3.2: Funktionen des Verkehrsnetzes 12](#_Toc487555079)

[Tabelle 3.1: Funktionen der Verkehrsregeln 13](#_Toc487555080)

[Tabelle 3.1: Funktionen der Verkehrsteilnehmer 14](#_Toc487555081)

[Tabelle 3.1: Funktionen der Getter und Setter - Ampelsteuerung 15](#_Toc487555082)

[Tabelle 3.1: Funktionen des RabbitMQ 18](#_Toc487555083)

# Aufgabenstellung

# Gesamtübersicht und Idee

Für die Umsetzung der Aufgabenstellung wurden die Aufgaben in sechs große Bereiche gespalten: GUI, Verkehrsnetz, Verkehrsteilnehmer, Verkehrsregeln, Ampelsteuerung und Rabbit MQ. Zu beachten ist der Punkt, dass die Ampelsteuerung extern zur Verfügung gestellt werden musste. Um eine Kommunikation zwischen Ampelsteuerung und Verkehrssimulation zu erreichen, wurde mit Windows Communication Foundation (WCF) gearbeitet. Hierbei ist die Ampelsteuerung der Server, welcher den Dienst der Ampel zur Verfügung stellt und die Verkehrssimulation der Client, welcher den Status der Ampeln abgefragt.

## GUI

In der hier vorliegenden Architektur wurde die GUI als eigene Komponente entwickelt. Grund hierfür war eine entkoppelte Benutzeroberfläche von der Simulationslogik anzubieten. Die GUI stellt lediglich die einzelnen Elemente der Simulation dar und gibt den Input des Benutzers an die darunterliegenden Logikkomponenten weiter.

### Klasse „Verkehrsteilnehmer“

Die Klasse Verkehrsteilnehmer kümmert sich um die korrekte Darstellung aller Autos/LKWs im Straßennetz. Mit Hilfe der bereitgestellten Funktion können Verkehrsteilnehmer hinzugefügt, gelöscht und ihre Position updatet werden.

### Klasse „Ampel“

Die Klasse Ampel setzt die Ampeln an geregelte Kreuzungen und bietet Funktionen an mit denen der Status der Ampeln verändert werden kann (Rot, Grün, Gelb, Blinkend)

### Klasse „Obstacle“

Diese Klasse übernimmt das Einzeichnen eines Hindernisses. Sie bietet dem Benutzer die Funktion über einen Mausklick ein Hindernis einzufügen

## Verkehrsnetz

Beschreibung

### Klasse „1“

Beschreibung

### Klasse „2“

Beschreibung

## Verkehrsregeln

Beschreibung

### Klasse „1“

Beschreibung

### Klasse „2“

Beschreibung

### Klasse „2“

Beschreibung

## Verkehrsteilnehmer

Beschreibung

### Klasse „1“

Beschreibung

### Klasse „2“

Beschreibung

### Klasse „2“

Beschreibung

## Ampelsteuerung

Die Ampelsteuerungskomponente hat die Aufgabe, Ampeln nach Belieben zu erstellen, aktiv/inaktiv zu schalten, Statusphasen manuell anzupassen und automatisch umzuschalten, wenn die Phasen jeweils vorüber sind. Realisiert wurde diese Komponente mit 2 Klassen:

### Klasse „Ampeln“

Diese Klasse symbolisiert eine einzelne Ampel. Mithilfe dieser Klasse ist das Erstellen von Ampeln ermöglicht worden. Funktionen und Variablen zu dieser Klasse sind in Abschnitt 3.5.1 ersichtlich.

### Klasse „Ampelsteuerung“

Die Klasse Ampelsteuerung ist für das Verhalten der Ampeln verantwortlich. Hier wird ebenso der WCF Server gestartet. In Kapitel 3.5.2 werden die verwendeten Funktionen und Variablen beschrieben.

## RabbitMQ

Beschreibung

### Klasse „1“

Beschreibung

### Klasse „2“

Beschreibung

### Klasse „2“

Beschreibung

# Dokumentation der Funktionalität der Verkehrssimulation

In diesem Sektor werden alle Funktionen aller Komponenten erklärt. Ebenso wird darauf eingegangen, wie alle Komponenten miteinander Funktionieren.

## GUI

Die Komponente „GUI“ besteht aus den Klassen: Verkehrsteilnehmer, Ampel und Obstacle (siehe Kapitel 2.1), sowie den Interfaces: IVerkehrsnetz und IObject. In diesem Abschnitt wird genauer auf diese eingegangen.

### Funktionen der Klasse Verkehrsteilnehmer

Die folgende Liste gibt die Funktionen der Klasse Verkehrsteilnehmer wieder und dem zugehörigen „ObjectHandler“ wieder.

|  |  |
| --- | --- |
| Funktion | Erklärung |
| Verkehrsteilnehmer() | Konstruktor: Erzeugt entweder ein Auto oder einen LKW |
| update(int x, int y, int type, int direction) | Aktualisiert die Position/Typ/Ausrichtung der Verkehrsteilnehmers |
| getShape() | Gibt die Form (Rechteck, Quadrat) wieder |
| setColor(Color c) | Setzt die Farbe (Differenzierung zwischen den Gruppen) |
| getID() | Gibt die Id wieder |
| enter(obj sender, arg e) | Erkennt wenn der Verkehrsteilnehmer angeklickt wurde |
| leave(obj sender, arg e) | Erkennt wenn der Mausklick beendet wird |
| conaoleMsg(obj sender, arg e) | Schreibt eine Log-Massage mit der ID und Position des angeklickten Elements |

Die Attribute hierzu sind:

int id;  
 Shape shp;

int Xpos;

int Ypos;

int direction;

### Funktionen der Klasse ObjectHandler

Um die Darstellung der Verkehrsteilnehmer im Verkehrsnetz zu verwalten, muss ein Objekt Handler hinzugefügt werden. Dieser kümmert sich um alle Fahrzeuge/Hindernisse, die sich zurzeit auf der Map (Canvasoberfläche) befinden.

* publi bool addCarObject(int x, int y, int id);

Fügt ein Auto an den X/Y Koordinaten ein und liefert im Erfolgsfall ein „true“ zurück.

* publi bool addLKWObject(int x, int y, int id, int dir);

Fügt ein LKW an den X/Y Koordinaten ein.

* publi bool updateCarwithID(int x, int y, int id, int dir);

Aktualisiert die Position und Richtung eines Autos mit einer bestimmten Id.

* publi bool removeObject();

Entfernt das Objekt mit der zugehörigen ID von der Map.

* publi bool addObstacle(obj sender, Arg e);

Fügt ein Hindernis an dem angeklickten Punkt im Canvas ein.

### Funktionen der Klasse Ampel

Hier geht es lediglich um den Farbwechsel der Ampeln und das Einfügen an den geregelten Kreuzungen des Verkehrsnetzes. Die folgende Liste gibt die Funktionen der Klasse Ampel und dem zugehörigen Handler „AmpelHandler“ wieder.

|  |  |
| --- | --- |
| Funktion | Erklärung |
| Ampel(double x, double y, int dir, int id) | Konstruktor: Erstellt eine Ampel aus 4 Elementen mit bestimmter Position, ID und Ausrichtung |
| getObjId() / setObjId(int id) | Liefert die Id der Ampel zurück / Setzt die Id neu |
| getXPos() / setXPos(int x) | Liefert die x-Position der Ampel zurück / Setzt die x-Position neu |
| getYPos() / setYPos(int y) | Analog zur x-Postion |
| getGreenCircle() / setGreenCircle(Shape) | Liefert den grünen Kreis der Ampel / Setzt die ihn neu |
| getYellowCircle() / setYellowCircle(Shape) | Liefert den gelben Kreis der Ampel / Setzt die ihn neu |
| getRedCircle() / setRedCircle(Shape) | Liefert den roten Kreis der Ampel / Setzt die ihn neu |
| getObjShape / setObjectShape(shape) | Liefert den Rahmen der Ampel / Setzt die ihn neu |

Tabelle .: Funktionen der GUI

Die Attribute hierzu sind:

int status;  
 Shape redCircle;

Shape yellowCircle;

Shape greenCircle;

Shape shape;

int id;

bool green, yellow, red;

int xPos;

int yPos;

int direction;

### Funktionen der Klasse AmpelHandler

Um die Darstellung der Ampeln im Verkehrsnetz zu verwalten, muss ein Ampel Handler hinzugefügt werden. Dieser kümmert sich um die korrekte Darstellung der Ampeln und des jeweiligen Status.

* publi bool addTrafficLight(int x, int y, int id, int dir);

Fügt eine Ampel an den X/Y Koordinaten mit bestimmter Ausrichtung ein und liefert im Erfolgsfall ein „true“ zurück.

* publi void setGreen(int id);

Setzt die Ampel mit der übergegbenen Id auf grün.

* publi void setYellow(int id);

Setzt die Ampel mit der übergegbenen Id auf gelb.

* publi void setRed(int id);

Setzt die Ampel mit der übergegbenen Id auf rot.

* publi void setNext(int id);

Prüft welche Farbe die Ampel mit der Id anzeigt und schaltet auf die nächst-logische Farbe.

* publi void setStatus(int id, int status);

Setzt die Farbe der Ampel nach einem übergebenen Status.

### Funktionen der Klasse Obstacle

Die Klasse „Obstacle“ erzeugt ein Hindernis. Da, für die wenigen Funktionen der Klasse ein separater Handler keinen Sinn machen würde, wurden die Funktionen im „ObjectHandler“ implementiert

|  |  |
| --- | --- |
| Funktion | Erklärung |
| Obstacle(int id, int x, int y) | Konstruktor: Erzeugt ein neues Hindernis mit einer Id und Position im Canvas |
| getShape() | Gibt die Form des Hindernisses wieder |

Die Attribute hierzu sind:

int id;  
 Shape shp;

int Xpos;

int Ypos;

### Funktionen der Interfaces IObject

Dieses Interface dient als Schnittstelle zur Komponente: Verkehrsteilnehmer. Die folgende Tabelle zeigt die Funktionen des Interfaces:

|  |  |
| --- | --- |
| Funktion | Erklärung |
| addCarObject(int x, int y, int id) | Fügt ein Auto der GUI hinzu |
| addLKWObject(int x, int y, int id, int dir) | Fügt ein LKW der GUI hinzu |
| updateCarwithID(int x, int y, int id, int dir); | Aktualisiert die Position in der GUI |
| removeObject(int id) | Löscht ein Verkehrsteilnehmer |

### Funktionen der Interfaces IVerkehrsnetz

Dieses Interface dient als Schnittstelle zur Komponente: Verkehrsnetz. Die folgende Tabelle zeigt die Funktionen des Interfaces:

|  |  |
| --- | --- |
| Funktion | Erklärung |
| getGreenLight() / setGreenLight(Shape) | Liefert den grünen Kreis der Ampel / Setzt die ihn neu |
| getGreenLight() / setGreenLight(Shape) | Liefert den gelben Kreis der Ampel / Setzt die ihn neu |
| getRedLight() / setRedLight(Shape) | Liefert den roten Kreis der Ampel / Setzt die ihn neu |
| getXPos() / setXPos(int x),  getYPos() / setYPos(int y), | Liefert die Position der Ampel zurück / Setzt die Position neu |
| getObjId() / setObjId(int id) | Liefert die Id der Ampel zurück / Setzt die Id neu |

## Verkehrsnetz

Beschreibung

### Funktionen

Beschreibung

* Beispielfunktion 1

Beschreibung der Funktion

* Beispielfunktion 2

Beschreibung der Funktion

* Beispielfunktion 3

Beschreibung der Funktion

### Deklarierung der Funktionen

Beschreibung

|  |  |
| --- | --- |
| Funktion | Erklärung |
| Funktion 1 | Beschreibung 1 |

Tabelle 3.2: Funktionen des Verkehrsnetzes

## Verkehrsregeln

Beschreibung

### Funktionen

Beschreibung

* Beispielfunktion 1

Beschreibung der Funktion

* Beispielfunktion 2

Beschreibung der Funktion

* Beispielfunktion 3

Beschreibung der Funktion

### Deklarierung der Funktionen

Beschreibung

|  |  |
| --- | --- |
| Funktion | Erklärung |
| Funktion 1 | Beschreibung 1 |

Tabelle .: Funktionen der Verkehrsregeln

## Verkehrsteilnehmer

Beschreibung

### Funktionen

Beschreibung

* Beispielfunktion 1

Beschreibung der Funktion

* Beispielfunktion 2

Beschreibung der Funktion

* Beispielfunktion 3

Beschreibung der Funktion

### Deklarierung der Funktionen

Beschreibung

|  |  |
| --- | --- |
| Funktion | Erklärung |
| Funktion 1 | Beschreibung 1 |

Tabelle .: Funktionen der Verkehrsteilnehmer

## Ampelsteuerung

Die Ampelsteuerung besteht aus 2 Klassen, siehe Kapitel 2.5.1 und 2.5.2 sowie deren Methoden und Variablen. In diesem Abschnitt wird genauer auf diese eingegangen.

### Funktionen der Klasse Ampel

Damit die Phasendauer einer Ampel, der Ausfall einer Ampel oder die ID abgefragt werden kann, wurden Getter und Setter sowie variablen in der Klasse angelegt:

|  |  |
| --- | --- |
| Funktion | Erklärung |
| getStatus() | Holt Ampelstatus |
| getRotPhase() | Gibt Zeit der Rotphase wieder |
| getGelbPhase() | Gibt Zeit der Gelbphase wieder |
| getGruenPhase() | Gibt Zeit der Grünphase wieder |
| getDefect() | Gibt Defekt der Ampel an |
| getID() | Holt ID der Ampel |
| getSekundenzähler() | Erhält Sekundenzähler |
| setSekundenzähler(int value) | Setzt den Sekundenzähler |
| setRotPhase(int value) | Setzt die Rotphasendauer |
| setGelbPhase(int value) | Setzt die Gelbphasendauer |
| setGruenPhase(int value) | Setzt die Grünphasendauer |
| setDefect(int value) | Setzt eine Ampel Defekt |
| setStatus(int value) | Setzt den Status einer Ampel |
| setID(int value) | Setzt bei Erstellung die ID der Ampel |

Tabelle .: Funktionen der Getter und Setter - Ampelsteuerung

Die Variablen hierzu sind:

int sekundenzähler = 0;  
 int Status;

int ID;

bool defect = false;

int rotphase = 3;

int gelbphase = 1;

int gruenphase = 3;

### Funktionen der Klasse Ampelsteuerung

Damit die Ampelsteuerung für die Verkehrssimulation verfügbar ist und per WCF erreichbar ist, muss folgendes hinzugefügt werden:

[ServiceBehavior(Name = "Ampelsteuerung", InstanceContextMode = InstanceContextMode.Single)]

Hiermit gibt man die Ampelsteuerung unter dem Namen „Ampelsteuerung“ am Localhost bekannt und lässt somit den Zugriff auf diese zu. Weiters werden folgende Funktionen benötigt:

* private void StartServer();

Diese Funktion erstellt einen ServiceHost, welcher von der Verkehrssimulation zum Erstellen einer Verbindung benötigt wird. Weiters wird gewartet, bis von der Verkehrssimulation die Anzahl der benötigten Ampeln geliefert wird. Die Ampeln werden erstellt und ein Timer zum Schalten der Phasen wird pro Ampel gestartet.

* private Task HandleTimer();

Dieser Timer inkrementiert den Sekundenzähler aus der Klasse „Ampeln“ und erzeugt so einen Phasenwechsel.

* public int getAmpelStatus(int ampelid);

Diese Funktion erhält die eine AmpelID. Diese ID zeigt auf eine Ampel und gibt den Status dieser Ampel zurück. Wird „0“ anstelle der AmpelID mitgegeben, so erhält man den Status aller erstellten Ampeln.

* public string getAmpelAusfall(int ampelid)

Diese Funktion gibt Information darüber, ob eine Ampel ausgefallen ist aufgrund der mitgegebenen AmpelID.

* public void setAmpelAusfall(int ampelid)

Mit setAmpelAusfall ist man in der Lage, eine Ampel aufgrund dessen ID Auszuschalten.

* public void setAmpelOn(int ampelid);

Hier kann man eine Ampel mit mitgegebener ID wieder einschalten, sofern sie ausgeschaltet ist.

* public void setAmpelStatus(int ampelid, int neuerStatus);

Hier kann man den Status einer Ampel manipulieren bzw. angeben, ob eine Ampel Rot, Gelb, oder Grün ist.

* public string getRotPhase(int ampelid);

Hier erhält man aufgrund der ID die Zeit der Rotphase wieder.

* public string getGelbPhase(int ampelid);

Hier erhält man aufgrund der ID die Zeit der Gelbphase wieder.

* public string getGruenPhase(int ampelid);

Hier erhält man aufgrund der ID die Zeit der Grünphase wieder.

* public void setRotPhase(int ampelid, int zeit);

Hier kann man aufgrund der ID die Zeit der Rotphase einstellen.

* public void setGelbPhase(int ampelid, int zeit);

Hier kann man aufgrund der ID die Zeit der Gelbphase einstellen.

* public void setGruenPhase(int ampelid, int zeit);

Aufgrund der ID der Ampel wird hier die Zeit der Grünphase eingestellt.

* public void setAmpelAnzahl(int anzahl);

Diese Funktion wird zu Beginn ausgeführt, um die Anzahl der Ampeln zu erstellen. Diese Funktion wird von der Verkehrssimulation verwendet.

* public List<Ampeln> factory(int anzahl);

Diese Funktion erstellt die Ampeln.

* static void Main();

In der Main wird die Ampelsteuerung ausgeführt.

Alle Funktionen werden über das Interface „IAmpelService“ der Verkehrssimulation zur Verfügung gestellt.

### Variablen der Klasse Ampelsteuerung

Damit die Funktionen miteinander Funktionieren, wurden folgende Variablen benötigt:

ServiceHost host;

bool \_serverRunning = false;

bool run = false;

public static List<Ampeln> Trafficlights = new List<Ampeln>();

public static int Anzahl = 0;

static Timer Ampeltimer;

## RabbitMQ

Beschreibung

### Funktionen

Beschreibung

* Beispielfunktion 1

Beschreibung der Funktion

* Beispielfunktion 2

Beschreibung der Funktion

* Beispielfunktion 3

Beschreibung der Funktion

### Deklarierung der Funktionen

Beschreibung

|  |  |
| --- | --- |
| Funktion | Erklärung |
| Funktion 1 | Beschreibung 1 |

Tabelle .: Funktionen des RabbitMQ

# Zusätzliche externe Komponenten

Welche externen Komponenten kamen zum Einsatz?

# Zusammenfassung und Ausblick

Was wurde geschafft und was nicht, bzw. wo gibt’s noch kleine Probleme

## Derzeit nicht implementiert

## Ausblick