Verifikationsziele Uppaal-Modell „Autonome Taxibusse“

Inhaltsverzeichnis

Einführung **1**
Eckdaten **1**
Automaten **2**
Passagier 2
Taxi 3
Anmelde-Server 4
Koordinator-Server 5
Ziele **5**
Ergebnisse **5**
Auswertung **7**
Fazit **8**

Einführung

Für die Verifizierung des Uppaal-Modells wurden in Absprache mit Herrn Korf nachfolgende Ziele zur Verifikation festgelegt. Die Passagiere werden im Vorhinein über einen Generator in Java geschrieben generiert. Alle Tests wurden mit mehreren vorgenerierten „Sets“ an Passagieren mit folgenden Eckdaten geprüft.

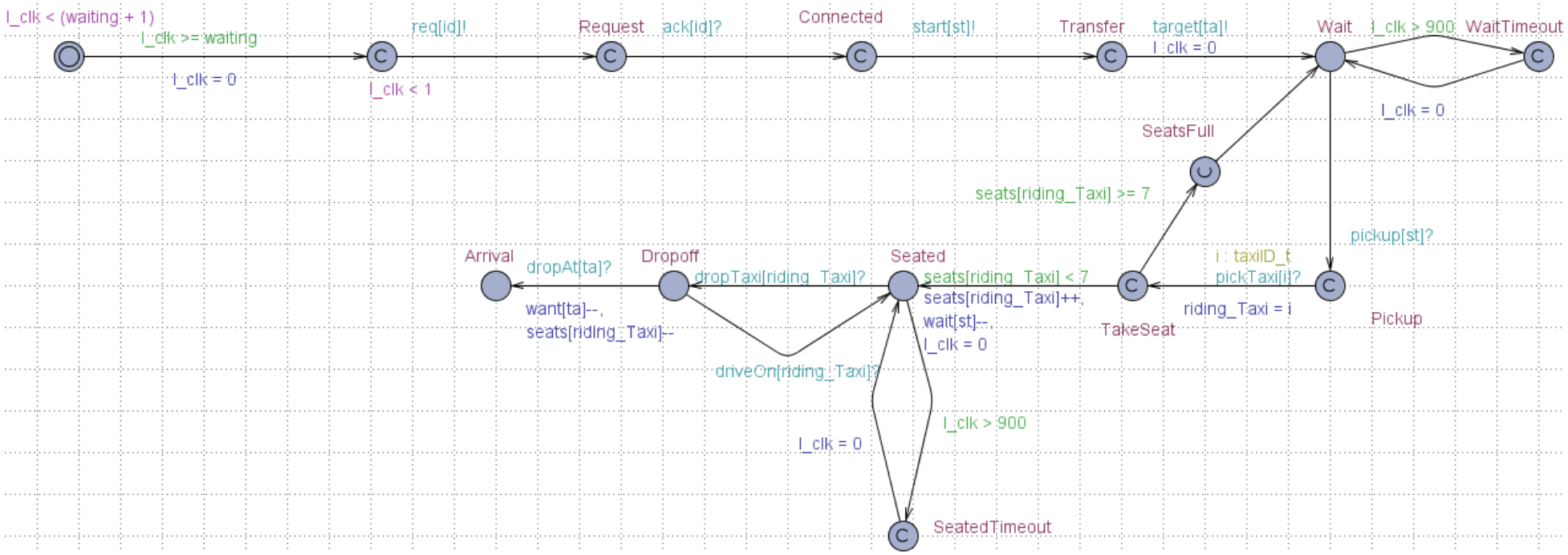
Eckdaten

Passagier: Anzahl = 20; l_clk = lokale clock; start/target zufällig; Anmeldung nach spätestens 400 ticks

Taxi: Anzahl = 1; Sitze = 7; l_clk = lokale clock; maximale Rundenzeit 890 ticks

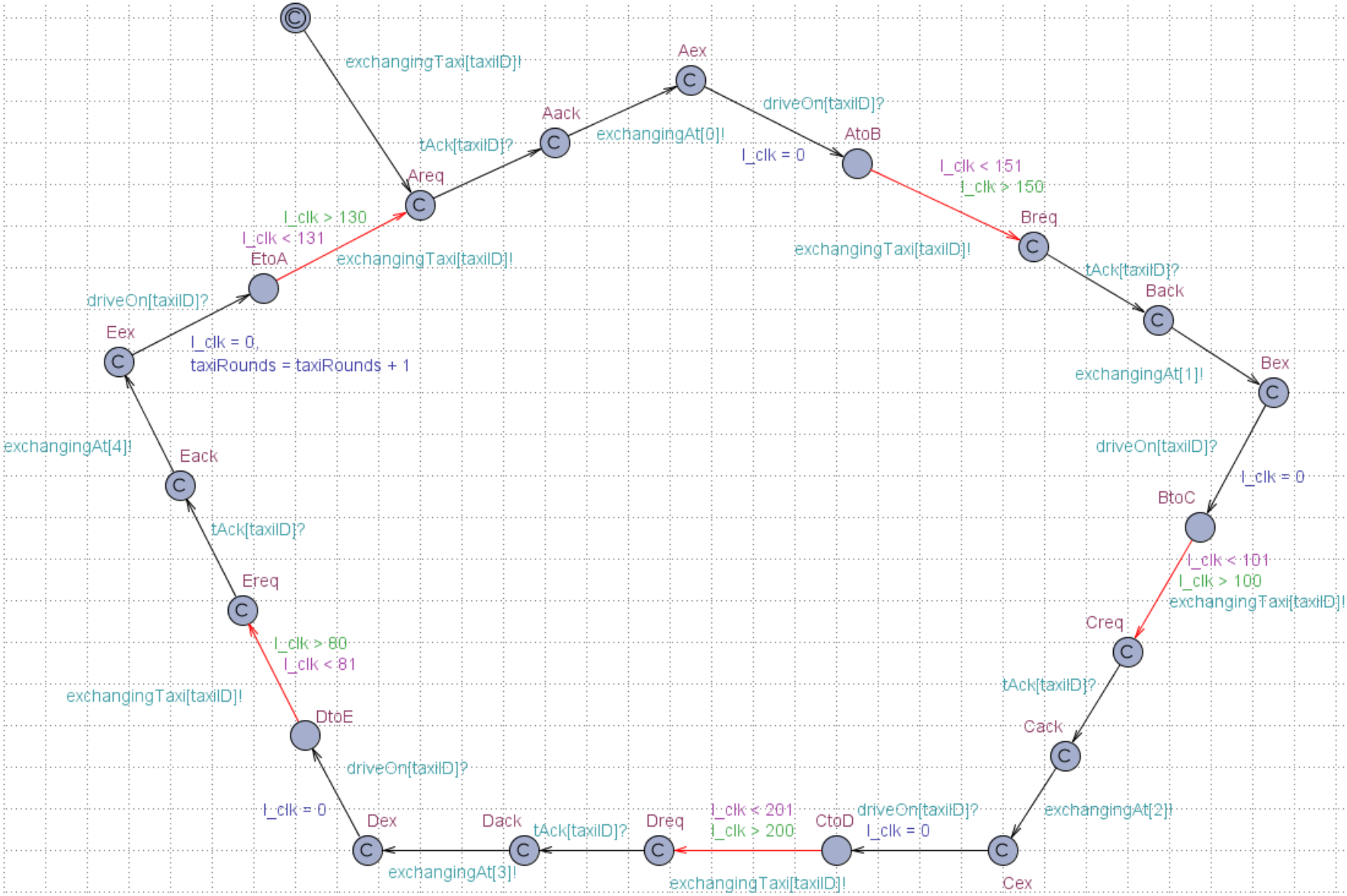
Automaten

Passagier



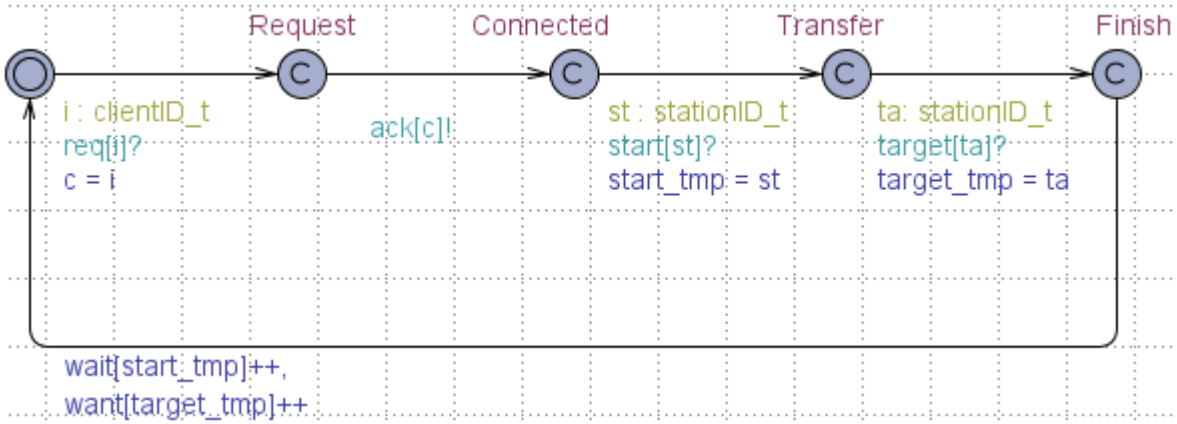
Passagiere sind mit dem obigen Automaten realisiert. Ein Passagier wartet eine gegebene (pseudo-zufällige) Zahl und meldet sich dann im System (Anmelde-Server) an. Danach gelangt er in den Zustand Wait, welcher einen Zeugen-Zustand WaitTimeout besitzt, der kurzzeitig betreten wird, wenn der Passagier eine bestimmte Zeit wartet, ohne dass ein Taxi ihn einsammelt. Wenn ein Taxi an seiner Station vorbeifährt, dann durchläuft der Passagier die Zustände Pickup und TakeSeat. Ist das Taxi bereits voll, so geht er in den Zustand Wait zurück und durchläuft dabei den Zeugen-Zustand SeatsFull. Hat das Taxi freie Plätze, dann gelangt er stattdessen in Seated, welcher auch einen Zeugen-Zustand SeatedTimeout besitzt, der kurzzeitig betreten wird, wenn der Passagier eine bestimmte Zeit im Taxi sitzt. Der weitere Pfad (Dropoff->Arrival) wird betreten, wenn das Taxi an der Ziel-Haltestelle hält. Hält das Taxi an einer anderen Haltestelle als das Ziel, dann geht der Passagier zurück in den Seated-Zustand.

Taxi


exchangingTaxi[taxiID]!
Aex
driveOn[taxiID]?
Aack
exchangingAt[0]!
tAck[taxiID]?
AtoB
l_clk = 0
l_clk < 151
l_clk > 150
Areq
l_clk > 130
l_clk < 131
EtoA
exchangingTaxi[taxiID]!
exchangingTaxi[taxiID]!
Breq
tAck[taxiID]?
Back
driveOn[taxiID]?
Eex
l_clk = 0,
taxiRounds = taxiRounds + 1
exchangingAt[1]!
Bex
exchangingAt[4]!
Eack
driveOn[taxiID]?
l_clk = 0
BtoC
tAck[taxiID]?
l_clk < 101
l_clk > 100
exchangingTaxi[taxiID]!
Ereq
Creq
l_clk > 80
l_clk < 81
tAck[taxiID]?
exchangingTaxi[taxiID]!
Cack
DtoE
driveOn[taxiID]?
exchangingAt[2]!
l_clk = 0
Dex
Dack
tAck[taxiID]?
Dreq
l_clk < 201
l_clk > 200
CtoD
driveOn[taxiID]?
l_clk = 0
exchangingAt[3]!
exchangingTaxi[taxiID]!
Cex

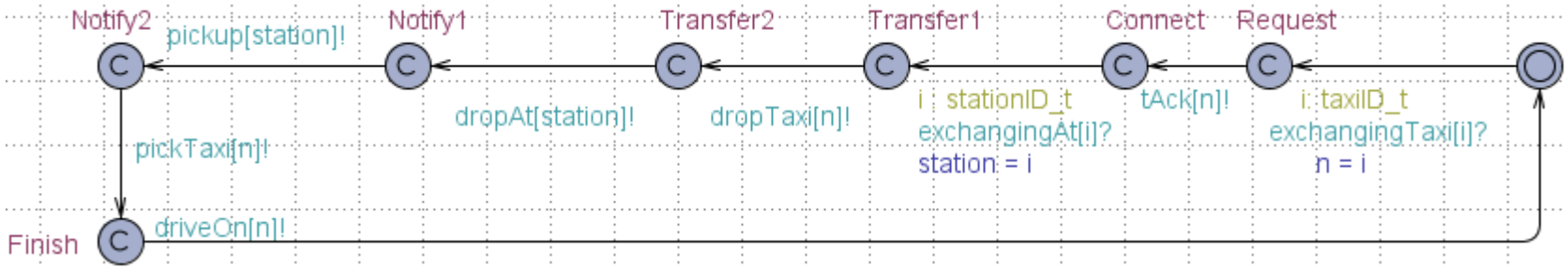
Das Taxi besteht aus Zuständen, die einerseits eine Rund-Route über 5 Haltestellen repräsentieren und andererseits die Kommunikation mit dem Koordinator-Server durchführen. Die Gefahrenen Routen sind die roten Transitionen mit jeweils unterschiedlichen Zeiten, die benötigt werden. Danach stellt das Taxi die Kommunikation mit dem Koordinator-Server her und im Zustand ‚X'ex findet der Austausch der Passagiere statt (Passagiere steigen aus -> neue Passagiere steigen ein), woraufhin der Koordinator-Server das Taxi befehligt, weiter zu fahren.

Anmelde-Server



Der Anmelde-Server besteht aus einem Request-Acknowledge Anmelde-Verfahren für die Passagiere mit nachfolgendem Austausch der Start- und Ziel-Halstestellen, welche dann in den beiden Arrays wait[MAX_STATION] und want[MAX_STATION] registriert werden. Die Indexe beider Arrays repräsentieren die Haltestellen mit den Nummern 0..4.

Koordinator-Server



Der Koordinator-Server geht Verbindungen mit Taxis ein, die einen Austausch an einer Haltestelle durchführen wollen. Daraufhin wird den Passagieren aus diesem Taxi mitgeteilt, dass ein Austausch stattfindet und an welcher Haltestelle. Die Passagiere führen ihre Aktionen daraufhin durch und dem Taxi wird signalisiert, dass es weiter fahren darf.

Ziele

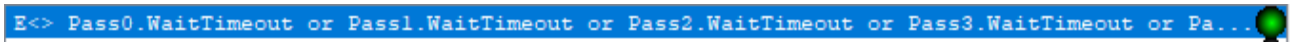
1. Maximale Wartezeit wird eingehalten
2. Maximale Gesamtfahrzeit wird eingehalten
3. Es bleibt kein Passagier für immer im Taxi sitzen
4. Es gibt mehr Passagiere, als das Taxi auf einmal befördern kann und alle kommen an.
5. Alle nehmen das erste Taxi, sofern Platz frei
6. Keiner steht im Taxi (es werden nicht mehr als die verfügbaren Plätze belegt)

Ergebnisse

1. Um zu verifizieren, dass kein Passagier eine bestimmte Wartezeit überschreitet, wurde eine Kante vom Wait-Zustand zu einem WaitTimeout-Zustand zugefügt, die mit einer Guard l_clk > 900 versehen wurde, was etwa einer Taxi-Runde entspricht. Die Verifikation findet konkret mit folgender Syntax statt:

   E<> Pass0.WaitTimeout or Pass1.WaitTimeout or Pass2.WaitTimeout or …

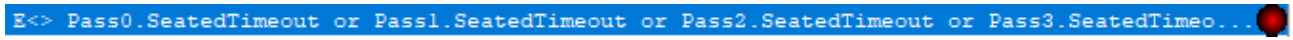
   Wertet die Formel wahr aus, dann gibt es mindestens 1 Passagier, der zu lange wartet. In diesem konkreten Testfall tut er dies aufgrund des gewählten Timeouts. Allerdings müssen 20 Passagiere mit einem 7-Sitze Taxi befördert werden, weswegen einige Passagiere mindestens eine Runde warten müssen, um einzusteigen.



2. Um zu verifizieren, dass kein Passagier eine bestimmte Gesamtfahrzeit überschreitet, wurde eine Kante vom Seated-Zustand zu einem SeatedTimeout-Zustand zugefügt, die mit einer Guard l_clk > 900 versehen wurde, was in etwa einer Rundfahrt entspricht. Die Verifikation findet konkret mit folgender Syntax statt:

   E<> Pass0.SeatedTimeout or Pass1.SeatedTimeout or Pass2.SeatedTimeout or …

   Wertet die Formel wahr aus, dann gibt es mindestens 1 Passagier, der sich länger als 1 Runde im Taxi befindet. Da er dabei mindestens 1-mal an seiner Ziel-Haltestelle vorbei kommt sollte dies nicht passieren. Das Ergebnis ist wie folgt:

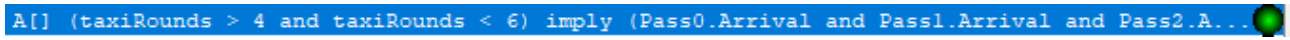


3. Um zu verifizieren, dass keiner ewig im Taxi sitzen bleibt reicht es, Punkt 2 mit einem Timeout >= Rundenzeit zu testen. Wenn kein Passagier sich länger als 1 Runde im Taxi befindet, dann sitzt auch keiner auf unbestimmte Zeit im Taxi fest.

4. Um zu verifizieren, dass bei einer Passagieranzahl größer als die Anzahl der Sitze alle Passagiere irgendwann ankommen, wurde ein kleiner Umweg gewählt, um Terminierung zu gewährleisten. Das Taxi erhöht jede Runde einmal eine Variable und zählt somit seine Runden. Nach ein paar Runden müssen alle Passagiere angekommen sein:

   A[] (taxiRounds > 4 and taxiRounds < 6) imply (Pass0.Arrival and Pass1.Arrival and …)

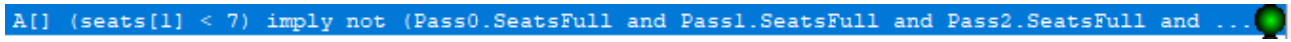
   Das Ergebnis lautet wie folgt:



5. Um zu verifizieren, dass alle Passagiere das erste Taxi nehmen, was an ihrer Haltestelle ankommt, wird die Kante betrachtet, die bei einem Austausch genommen wird, wenn bereits alle Sitze belegt sind (SeatsFull). Diese darf nicht genommen werden, wenn das Taxi freie Sitze hat. Die Formel lautet wie folgt:

   A[] (seats[1] < 7) imply not (Pass0.SeatsFull and Pass1.SeatsFull and …)

   Wertet die Formel wahr aus, dann nimmt jeder Passagier das erste Taxi falls ein Sitz frei ist. Das Ergebnis lautet wie folgt:



6. Um zu verifizieren, dass sich nicht mehr Passagiere sich im Taxi befinden, als es Sitze hat, wird getestet, ob die Anzahl an Passagieren im Taxi in irgendeinem Zustand größer ist als die maximalen Sitze:

   A[] seats[1] <= 7

   Wertet die Formel wahr aus, so sind nie zu viele Passagiere in einem Taxi. Das Ergebnis lautet wie folgt:

Auswertung

Unter den o.g. Zielen befinden sich harte und weiche Ziele. Harte Ziele sind Randbedingungen des CPS und dürfen nicht verletzt werden, da sie ansonsten einen Fehler-Zustand des Gesamtsystems bezeugen. Weiche Ziele hingegen sind von den Eckdaten des Systems abhängig und dürfen verletzt werden, ohne einen Fehler im System zu erzeugen. Werden diese Ziele allerdings verletzt, dann spricht es für eine degradierte Qualität des Service.

Zu den harten Zielen gehören unter anderem die Ziele 3-6. Die Gründe hierfür sind wie folgt:

Ziel 3: Es bleibt kein Passagier für immer im Taxi sitzen
Dieses Ziel ist selbsterklärend. Ein Passagier benutzt das Taxi nur, um sein Ziel zu erreichen. Das Taxi fährt mindestens ein Mal in einer Runde an diesem Ziel vorbei und der Passagier ist verpflichtet hier auszusteigen, da der Service beendet ist. Das Ziel wurde mit unterschiedlichen Sets mit 10-30 Passagieren und wertet stets wahr aus.

Ziel 4: Es gibt mehr Passagiere, als das Taxi auf einmal befördern kann und alle kommen an.
Da es vorkommen kann, dass die Kapazitäten des Service zu Stoßzeiten nicht ausreichen, um alle Passagiere beim ersten mal einzusammeln sollte es dem Taxi trotzdem möglich sein, die zurückgelassenen Passagiere bei der nächsten Rundfahrt einzusammeln. Abgesehen davon, dass dies möglich sein muss, um ein fehlerfreies System zu ermöglichen, bedeutet eine unausgewogene Taxi-Passagier Beziehung eine Degradierung der Qualität des Service, weshalb die Zahl der Taxis an die Zahl der Passagiere bei laufendem Betrieb angepasst werden sollte.

Ziel 5: Alle nehmen das erste Taxi, sofern Platz frei
Ist ein Passagier für eine Route angemeldet und ein Taxi erscheint bei ihm, dann muss er dieses nehmen, sofern Platz ist. Ist dies nicht der Fall, dann entspricht dies in der Realität für eine nicht-Inanspruchnahme des Dienstes, woraufhin der Passagier aus dem System entfernt wird.

Ziel 6: Keiner steht im Taxi (es werden nicht mehr als die verfügbaren Plätze belegt)
Es ist gefordert, dass nicht mehr Passagiere ein Taxi nehmen, als dieses Plätze hat, da dies in der Realität nicht legal ist.

Zu den weichen Zielen gehören Ziele 1 und 2. Die Gründe hierfür sind wie folgt:

Ziel 1: Maximale Wartezeit
Je nachdem, was für eine Zeit gewählt wird, kann dieses Ziel falsch auswerten. Ist die Zeit z.B. mit einer Rundenzeit belegt und es gibt zu viele Passagiere, dann kann es passieren, dass die Wartezeit von einigen Passagieren überschritten wird. Folgen sind Unzufriedenheiten der Kunden, welche mit Vergünstigungen oder angebotenem Abbruch des Service bekämpft werden könnten. Wenn möglich sollte die Anzahl der Taxis so gewählt sein, dass dies nicht passiert.

Ziel 2: Maximale Fahrzeit
Je nachdem, was für eine Zeit gewählt wird, kann dieses Ziel falsch auswerten. Wird die Zeit überschritten ist entweder ein zu enger Zeitraum gewählt oder es ist ein Indiz für eine schlechte Verkehrssituation, die dafür sorgt, dass vorgesehene Fahrzeiten nicht eingehalten werden können.

Fazit

Die harten Ziele müssen bei der Programmierung der Systeme berücksichtigt werden. Weiterhin ist aufgrund der weichen Ziele zu beachten, dass Taxis, als auch Passagiere skalierbar sein müssen. Grundsätzlich ließe sich die Kommunikation zwischen Passagier und Taxi auch direkt lösen, es ist also auch möglich eine dezentrale Struktur zu realisieren. Ob zentral oder dezentral gewählt werden sollte hängt mitunter von der Skalierung des Gesamtsystems ab. Nimmt die Zahl der Taxis und Passagiere enorm zu müssen auch mehr Server in Betrieb genommen werden, was sich vor allem dann bemerkbar macht, wenn der Service in andere Länder expandiert. Dafür ist die Architektur mit einem koordinierenden Server eventuell einfacher zu realisieren.

Davon abgesehen besteht die Anforderung an die Kommunikation zwischen den Systemen, dass sie leichtgewichtig ist, was vor allem für mobile Endgeräte gilt. Zusätzlich sollten viele Teilnehmer möglich sein und eine sichere zuverlässige Verbindung möglich sein, die geringe Verzögerungen erlaubt und auch mehrere Sekunden Verbindungsverlust bewältigt. Diese Anforderungen werden vor allem von MQTT erfüllt.