SEIBT & STRAUB
INGENIEURBÜRO

GefoS

optiTAX Connect

DRAFT

Autoren: Markus Becker, Seibt & Straub GmbH
Matthias Bremer, GefoS mbH

Stand: Februar 2013

SEIBT & STRAUB
INGENIEURBÜRO

GefoS

Inhaltsverzeichnis

1 Einleitung

optiTAX Connect ist eine Schnittstellenbeschreibung.

Die Schnittstelle dient der Übermittlung von Fahraufträgen und damit verbundenen Informationen zwischen Bestellsystemen und Flottenmanagementsystemen verschiedener Hersteller.

Damit soll eine Verbindung zwischen (Smartphone) Bestellsystemen und (Taxi) Vermittlungssystemen geschaffen werden.

Um dieses Ziel zu erreichen wird folgende grundlegende Struktur vorgeschlagen:
Alle beteiligten Systeme koppeln an einen gemeinsamen Vermittler. Dieser Vermittler kommuniziert von den angekoppelten Bestellsystemen generierte Fahraufträge zu den angeschlossenen Fahrtenvermittlungssystemen. Zur Klarstellung sei angemerkt, dass dieser Vorschlag die Server-Kopplung im Fokus hat – nicht die **direkte** Verbindung zu einzelnen Teilnehmern (Smartphone APPS, etc).

Die Implementation der Schnittstelle soll für alle beteiligten Systeme einfachst möglich sein. Deshalb werden ausschließlich frei zugängliche Standards eingesetzt.

Die Schnittstelle zu den Bestellsystemen und die Schnittstelle zu den Dispositionssystemen nutzen identische Befehle und Datenstrukturen.

2 Allgemeine Festlegungen

Die Schnittstellen sollen keine unnötigen Freiheitsgrade beinhalten, die die Implementation und die Tests erschweren. Deshalb werden folgende grundsätzlichen Vereinbarungen getroffen:

- Zeichensatz UTF-8
- Objekte werden in Json Notation übermittelt
- Transport der Objekte mittels http(s) oder Telnet-like

Für den Transport der Objekte werden zu diesem Zeitpunkt zwei Vorschläge gemacht. Bis zur ersten gültigen Fassung des Dokuments muss entschieden werden, ob einer oder eventuell auch beide Varianten Teil der Beschreibung werden. Serverseitig sollte die Implementation beider Varianten keinen erheblichen Overhead darstellen, für die angeschlossenen Systeme könnte es, je nach verwendeter Technologie einen Vorteil darstellen, die Transportschicht frei wählen zu können.

2.1 Festlegungen zur Darstellung in diesem Dokument

Die Konventionen der Java Naming Convention (siehe http://www.oracle.com/technetwork/java/codeconv-138413.html) werden zu Grunde gelegt. Das sollte keinerlei Einschränkungen für andere Umgebungen darstellen und ist der Basis von Json aus dieser Umgebung geschuldet.

Bezeichnung von Typen und Objekten: Für primitive Json Datentypen werden die Bezeichnungen der Json Definition (http://www.json.org/) benutzt.
Für Objekttypen wird CamelCase wie in Java verwendet. Typbezeichungen werden **fett**, Feldbezeichnungen *kursiv* geschrieben.

2.2 http(s) Transport

Der Client initiiert die logische Verbindung mittels http(s) GET. Dabei übergibt er in den Parametern Benutzer und Passwort. Der Server hält die Verbindung zum Client offen (Transfer-Encoding chunked). Über diese Verbindung wird eine Session ID an den Client übermittelt. Danach sendet der Server über diese Verbindung Ereignis-Objekte an den Client.

Der Client übermittelt Anfragen (Funktionsaufrufe) und Antworten auf Ereignisse per http(s) POST an den Server. Dabei übergibt er seine Session-ID und ein JSON Objekt als Parameter und erhält ein Antwort Objekt vom Server.

2.3 Telnet-like Transport

Der Client öffnet eine TCP (ssl) Verbindung zum Server und führt ein Benutzer/Passwort Login durch. Danach werden über die Verbindung Json Objekte zwischen Client und Server übermittelt. Dabei werden die Objekte wie beim http(s) chunked Transfer-Encoding übermittelt.

2.4 Chunked Transfer-Encoding

Die übermittelten Daten bestehen aus einer Folge folgender Sequenz:
<Länge>CRLF<Objekt>CRLF. Die Folge wird mit CRLF abgeschlossen.

<Länge> ist die Anzahl von Bytes des Objekts als ASCII-Hex Wert. Beispiel: Das Objekt enthält 17 Zeichen. Dann hat <Länge> den Wert '11'. Falls das Objekt 123456 Zeichen enthält, wird für <Länge> '1E240' übermittelt.

Siehe RFC2616, Section 3.6.1 (http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html), dort ist das genaue Format erklärt.

optiTAX Connect verwendet keine Extensions. Dennoch sollten die Parser exakt RFC2616, 3.6.1 respektieren und implementieren.

2.5 Funktionsaufrufe und Ereignisse

Ereignisse, also Informationen vom Server zum Client werden wie Funktionsaufrufe des Clients auf dem Server gehandhabt – lediglich mit vertauschten Rollen. Der Client beantwortet auch Events, so dass auch bei Verbindungsabbrüchen verlustfrei übermittelt werden kann.
In dieser Dokumentation werden alle Vorgänge aus Sicht der Bestellsysteme beschrieben – wobei die Schnittstellen zu den Vermittlungssystemen identische Datenstrukturen verwenden.

Beispiel:
Ein angeschlossenes Bestellsystem generiert einen Fahrauftrag. Es löst die **Funktion** placeOrder des Vermittlers auf. Der Vermittler entscheidet über ein geeignetes Vermittlungssystem und generiert in Richtung dieses Systems das **Ereignis** placeOrder
In dieser Dokumentation werden die Strukturen für placeOrder – aus Sicht des Bestellsystems, also unter der Liste der Funktionen geführt.

Zu dieser Beschreibung gehört auch die Dokumentation der Verantwortlichkeiten für die Dateninhalte.

Beispiel:
Ein Bestellsystem ruft placeOrder auf. Der Vermittler generiert eine eindeutige Auftragsnummer

orderID und ein placeOrder Ereignis an einem angeschlossenen Vermittlungssystem. Das Vermittlungssystem akzeptiert den Auftrag. Daraufhin fügt der Vermittler die sysID des beauftragten Systems dem Auftrag hinzu und beantwortet die Anfrage gegenüber dem Bestellsystem entsprechend.

2.6 Teilnehmende Systeme

Alle Systeme, die Verbindungen zum Server aufbauen, erhalten eine eindeutige Kennzeichnung (sysID), Benutzername und ein Zugangspasswort.

Beispiele:
Kennungen für Vermittlungssysteme: HANNOVER_HALLO_TAXI3811_OPTITAX, DARMSTADT_TAXIFUNK_OPTITAX,...
Kennungen für Bestellsysteme: CAB4ME, TAXI_BREMEN_APP, DEIN_TAXI, OPTITAX_TR_GPRS,...

Die Kennungen werden vom Betreiber des Vermittlers ausgegeben.

Bestellsysteme können die Auftragsverarbeitung durch bestimmte Systeme gezielt anfordern oder die Entscheidung über die Auswahl dem Vermittler überlassen. Die damit verbundenen Mechanismen und Vorgehensweisen sind nicht Teil dieser Schnittstellenbeschreibung.

Eingehende Aufträge werden mit der sysID des Bestellsystems gekennzeichnet. Zusätzlich sollte eine – innerhalb des jeweiligen Bestellsystems eindeutige – Information über das Endgerät/Terminal/Benutzer des Bestellsystems vorhanden sein, um eine Kommunikation zwischen Vermittlungssystem und Besteller zu ermöglichen.

Seitens des Vermittlungssystems sollten – innerhalb dieses Vermittlungssystems eindeutige – Informationen über den ausführenden Teilnehmer angefügt und mitgeteilt werden.
Diese Informationen sind sowohl für die Kommunikationsvorgänge als auch für mögliche Geschäftsmodelle relevant.

2.7 Json

Objekte werden in Json Notation übermittelt. Als Zeichensatz ist UTF-8 vereinbart.

2.7.1 Zeit-und Datumsangaben

Json beinhaltet keine Festlegungen zur Übermittlung von Zeitangaben, Datumsangaben und Zeitintervallen. Deshalb sind für optiTAX connect spezielle Festlegungen nötig.

- für Zeit-Datumsangaben werden Unix Zeitstempel verwendet. (http://en.wikipedia.org/wiki/Unix_time)
- Für Zeitintervalle werden ganze Zahlen verwendet, die für die Anzahl Sekunden im Intervall stehen.

Als Bezeichnung für den Typ eines Zeitwertes wird in diesem Dokument **Date** vereinbart.
Unix Zeitstempel sind einfach zu implementieren, haben aber den Nachteil der schlechten Lesbarkeit, und eines der Ziele dieser Schnittstellendefinition ist es, die übermittelten Informationen möglichst auch einfach lesbar (human readable) darzustellen. Deshalb wird empfohlen, dass Implementationen (auf der Server-Seite) auch ISO8601 Formate (http://de.wikipedia.org/wiki/ISO_8601) unterstützen.
Vorschläge:

- Falls ein Zeitfeld in einer Zeichenkette übermittelt wird, werden diese als ISO8601 Zeitangaben behandelt, falls das Zeitfeld als Zahl übermittelt wird, handelt es sich um

einen Unix Timestamp.
- Server übermitteln Zeiten als Unix-Timestamp, um die Interpretation (parsing) auf den Client-Systemen zu vereinfachen.
- Zusätzlich können Server informativ Zeitfelder im ISO8601 Format ausgeben. Dazu werden zusätzliche Felder mit der Namenserweiterung „ISO“ verwendet. Beispiel: orderTime=nnnnnn, orderTimeISO=“2009-06-30T18:30:00*+02:00*“.

Hinweise zu Angaben im ISO8601 Format:
Die Angaben sollen die Zeitzone einschließen. Falls keine Zeitzone angeben wird, verwendet der Server seine Zeitzone und die Angaben können fehlerhaft interpretiert werden. Falls der Server Zeitangaben im ISO8601 Format ausgibt, verwendet er seine Zeitzone (Spätere Protokollversionen können die Zeitzone des Clients berücksichtigen).

3 Basisobjekte

Folgende Objekte werden universell vereinbart

3.1 Addr

Feld	*Typ*	*Beschreibung*
name	**string**	Name/Firma
hnr	**string**	*Hausnummer*
str	**string**	Straßenname
town	**string**	Postalischer Ort
pCode	**string**	Postleitzahl, optional
country	**string**	Länderkennzeichen, Deutschland=049, optional
hint	**string**	Anfahrtshinweis, optional
pos	**GeoPos**	Geografische Position, optional (siehe 3.3)
info	**string**	Hinweistext, optional
phone	**string**	Rufnummer, optional

3.2 Caller

Feld	*Typ*	*Beschreibung*
callerID	**string**	ID des Bestellgeräts, z.B. IMEI, falls vorhanden
callerTyp	**string**	*Art der ID (IMEI, ANDROID_ID, PHONE_NUMBER, COOKIE)*
sysID	**string**	sysID des Bestellsystems (siehe 2.6)

3.3 GeoPos

Geografische Position. Die Positionsangabe kann später weitere optionale Felder enthalten (Geschwindigkeit etc.)

Feld	*Typ*	*Beschreibung*
lon	**number**	Geografische Länge, WGS84, Dezimalgrad
lat	**number**	Geografische Breite, WGS84, Dezimalgrad
src	**string**	'GSM', 'GPS', 'MAP' Quelle der Position, optional
prec	**number**	Geschätzte Genauigkeit in Meter, Ganzzahl, optional

3.4 Beispiel:

{“lon“:9.123,“lat“:48.123,“src“:“GPS“,“prec“:50}

3.5 Order

Feld	*Typ*	*Beschreibung*
orderID	**string**	Eindeutige ID der Fahrt, falls vorhanden
srcID	**string**	SYS_ID des Bestellsystems (siehe 2.6)
snkID	**string**	SYS_ID des Vermittlungssystems (siehe 2.6)
srcRef	**Caller**	Angabe zum Teilnehmer des Bestellsystems
snkRef	**Vehicle**	Angabe zum beauftragten Teilnehmer des Vermittlungssystems (siehe 3.9)
depart	**Stop**	Anfahrbare Adresse, Beginn der Fahrt (siehe 3.7)
route	**Stops**	Abholpunkte, Letzter Eintrag: Ziel, optional (siehe 3.8)
opts	**OrderOpts**	Optionen für die Bestellung (siehe 3.6)

3.6 OrderOpts

[string] - Array mit zulässigen Bestelloptionen (siehe 4)

3.7 Stop

Feld	*Typ*	*Beschreibung*
loc	**Addr**	Anfahrbare Adresse (siehe 3.1)
arrivalTime	**Date**	vorgesehene Ankunftszeit laut Bestellung oder Fahrplan, optional
persons	**number**	Personenzahl laut Bestellung, optional
arrivalEstimate	**Date**	durch das Vermittlungssystem geschätzte Ankunftszeit, optional

3.8 Stops

[Stop] - Array mit **Stop**

3.9 Vehicle

Teilnehmendes Fahrzeug einer Flotte. *vehicleID* muss eindeutig in einer Flotte sein. *vehicleID*+*sysID* müssen systemweit eindeutig sein.

Feld	*Typ*	*Beschreibung*
vehicleID	**string**	Eindeutige Bezeichnung (z.B. Seriennummer des Funkgeräts)
sysID	**string**	sysID der Flotte (siehe 2.6)
radioID	**string**	Funkrufnummer/Konzessionsnummer zur Anzeige, optional
plate	**string**	Kennzeichen des Wagens zur Anzeige, optional

3.10 VehiclePos

Position eines Teilnehmers zu einem Zeitpunkt.

Feld	*Typ*	*Beschreibung*
vehicle	**Vehicle**	ID des Teilnehmers (siehe 3.9)
pos	**GeoPos**	Geografische Position zum Zeitpunkt *posTime* (siehe 3.3)
posTime	**Date**	Zeitpunkt der Abtastung
mobState	**string**	'AVAILABLE','BUSY','OUT_OF_SERVICE', optional

Beispiel:

4 Bestelloptionen

Es wird eine Liste mit zulässigen Optionen als Teil der Schnittstellenbeschreibung festgelegt. Falls ein Vermittlungssystem eine Option nicht kennt, kann es diese Option als informativen Text betrachten, oder den Auftrag ablehnen.

Beispiele:

KOMBI	Kombi-Fahrzeug
ENGLISH	Englisch sprechenden Fahrer
FRENCH	Französisch sprechenden Fahrer
PAY_CREDIT	Kreditkartenzahlung
PAY_CHIP	Bezahlung per Chip/PIN

…

5 Funktionsaufrufe

Funktionen enthalten den Namen der Funktion *func*, eine Sequenznummer *seq* und die Parameter in ein JSON Objekt gekapselt. Die Gegenstelle beantwortet den Aufruf mit einem Ergebnisobjekt. Falls dieses Ergebnisobjekt nicht innerhalb einer Maximalzeit eingeht, sollte die Verbindung neu initiiert werden und die Funktion mit derselben Sequenznummer *seq* nochmals aufgerufen werden. Scheitert der Aufruf wiederholt, bedeutet das einen Fehler der Kommunikation (Dienst steht nicht bereit)

Falls *seq* nicht angegeben wird, verwirft die Gegenstelle das Ergebnis. Es werden dann keine Antworten (auch keine Fehler) zurückgemeldet. Beispiel: Übermittlung von Fahrzeugpositionen, die nur für einen kurzen Zeitraum gültig sind, Einsparung von Transportvolumen.

Beispiel:
{“FUNC“:“NOP“,“SEQ“:1}
{“RES“:1,“INFO“:“Server is up since 12345h, 34 active Sessions, Time is 16:44:30+02“}

Fehlerbeispiel:
{“FUNC“:“READ_MAIL“,“SEQ“:2}
{“RES“:2,“ECODE“:1,“EMSG:“NOT implemented. This server can not READ_MAIL by now“}

5.1 Fehlerliste-globale Fehler

0	Kein Fehler
1	Funktion ist nicht implementiert
2	Formatfehler. Fehler beim parsen.

5.2 nop

Die Funktion kann jederzeit an die Gegenstelle gesendet werden und muss beantwortet werden. (Keep-Alive, Watchdog, Verbindungskontrolle)

Feld	*Typ*	*Beschreibung*
func	**string**	nop
seq	**number**	*eindeutige Sequenznummer des Aufrufs*
Antwort		
res	**number**	== Sequenz *seq* des Aufrufs
eCode	**string**	Fehlercode, optional
eMsg	**string**	Klartextmeldung bei Fehler, optional
info	**string**	Infomeldung optional

5.3 placeOrder

Feld	*Typ*	*Beschreibung*
func	**string**	placeOrder
seq	**number**	*eindeutige Sequenznummer des Aufrufs*
order	**Order**	Auftrag (siehe 3.5)
prefSys	**string**	sysID des bevorzugten Vermittlungssystems (siehe 2.6)
routingHint	**string**	Hinweis für die Auftragsverteilung/Weiterleitung an ein Vermittlungssystem (STRICT: Das angegebene System muss verwendet werden, LOOSE: Das angegeben System wird bevorzugt)
Antwort		
res	**number**	== Sequenz *seq* des Aufrufs
eCode	**string**	Fehlercode, optional
eMsg	**string**	Klartextmeldung bei Fehler, optional
info	**string**	Infomeldung optional
order	**Order**	Bestellung, incl. *orderID* und *snkID, arrivalEstimate* (siehe 3.5)

Die Verarbeitung der Auftrags (Übermittlung an ein Dispositionssystem) kann einige Zeit benötigen. Gegebenenfalls müssen mehrere Vermittlungssysteme angefragt werden und deren Antwort abgewartet werden, bevor eine Antwort erfolgt. Dennoch wird hier eine synchrone Verarbeitung vorgesehen (zur Diskussion)

Mögliche Fehler:

5.3.1 1001 Kein Provider gefunden

1002 INSERT durch Provider abgelehnt
1003 order.*srcRef* Blacklisted

5.4 queryOrder

Verfügbarkeitsanfrage. Parameter identisch mit *placeOrder*. Im Ergebnis wird eine geschätzte Ankunftszeit (*arrivalEstimate*) genannt.
Bestellsysteme sollte keinesfalls eine Verbindlichkeit der Antwort auf queryOrder erwarten - ein auf queryOrder folgendes placeOrder kann eine abweichende Bedienzeit liefern oder sogar abgelehnt werden. Spätere Versionen dieser Schnittstellenbeschreibung könnten eine verbindliche Verfügbarkeitsanfrage (Reservierung eines konkreten Wagens) beinhalten.

5.5 queryVehicles

Verfügbarkeitsanfrage mit Positionsmeldungen. Parameter identisch mit *placeOrder*. Im Ergebnis werden für die Fahrt mögliche Wagen genannt.

Antwort		
res	**number**	== Sequenz *seq* des Aufrufs
eCode	**string**	Fehlercode, optional
eMsg	**string**	Klartextmeldung bei Fehler, optional
info	**string**	Infomeldung optional
vehiclePos	**[VehiclePos]**	Array mit Wagenpositionen (siehe 3.10)

5.6 Ereignisse

Die Übermittlung von Informationen ausgehend von Vermittlungssystemen an Bestellsysteme wird Ereignis genannt, da die Schnittstelle aus Sicht des Bestellsystems beschrieben wird. Beispiel: Sobald ein Vermittlungssystem einen Auftrag an einen Wagen übermittelt hat, übermittelt es die entsprechenden Informationen an den Vermittler ({"func":"orderState",...}), dieser überträgt das Objekt weiter an das zuständige Bestellsystem.
Der Begriff "Ereignis" beschreibt in diesem Zusammenhang die generelle Richtung des Informationsaustauschs (Vermittlungssystem-->Bestellsystem).
Es ist zu beachten, dass zwischen Bestellsystem und Vermittlungssystem nur in einzelnen Fällen eine einfache 1:1 Zuordnung besteht. In vielen Fällen ("Radar" Fahrzeuganzeige) verarbeitet der Vermittler die Daten der Vermittlungssysteme weiter und stellt den Bestellsystemen konsolidierte Informationen (auf Anfrage) bereit.

5.7 orderState

Feld	*Typ*	*Beschreibung*
func	**string**	orderState
seq	**number**	eindeutige Sequenznummer des Aufrufs
orderID	**string**	ID der Bestellung
orderState	**string**	'inserted','dispatched','completed','cancelled'
mobileID	**Vehicle**	Beauftragter Wagen (siehe 3.9)
msg	**string**	Mitteilung an den Besteller „Wagen 123 hat die Bestellung...
msgTyp	**string**	'INFO','WARNING','ERROR', optional
pos	**GeoPos**	Aktuelle Position des Wagens, optional (siehe 3.3)

5.8 infoMsg

Feld	*Typ*	*Beschreibung*
func	**string**	infoMsg
seq	**number**	eindeutige Sequenznummer des Aufrufs
srcSysID	**string**	sysID (des Vermittlungssystems)
destSysID	**string**	sysID (des Bestellsystems)
callerID	**string**	Teilnehmer der die Nachricht erhalten soll (siehe 3.2)
msg	**string**	Nachricht
msgTyp	**string**	'INFO','WARNING','ERROR', optional

6 Funktionen für Vermittlungssysteme

6.1 updVehicles

Stammdatenupdate für teilnehmende Wagen. *vehicleID* muss angegeben werden. Falls *sysID* angegeben wird, muss der Wert mit der *sysID* des Vermittlungssystems übereinstimmen. Falls *radioID* und *plate* leer sind, wird der Wagen aus der Datenbank des Vermittlers entfernt.
Die Verwendung dieser Funktion ist für Vermittlungssysteme optional. Die Stammdaten können auch bei updVehPos angegeben werden.
Das Update von Stammdaten durch updVehicles und von Bewegungsdaten/Status durch updVehPos wird empfohlen.

Feld	*Typ*	*Beschreibung*
func	**string**	updVehicles
seq	**number**	eindeutige Sequenznummer des Aufrufs
vehicles	**[Vehicle]**	Array mit teilnehmenden Wagen (siehe 3.9)

6.2 updVehPos

Bewegungsdatenupdate. Die Felder *plate* und *radioID* müssen nicht angegeben werden, falls die Stammdaten vorher via updVehicles übermittelt wurden.

Feld	*Typ*	*Beschreibung*
func	**string**	updVehPos
seq	**number**	eindeutige Sequenznummer des Aufrufs
vehicles	**[VehiclePos]**	Array mit Wagenpositionen (siehe 3.10)

7 ToDo

In diesem Abschnitt sei kurz auf bereits bekannte Fragestellungen hingewiesen, die bis zur ersten gültigen Fassung noch beantwortet und in die Beschreibung eingearbeitet werden müssen.

Zentralenverzeichnis
Es muss eine Funktion definiert werden, über die Bestellsysteme geografisch „zuständige“ Vermittlungssysteme ermitteln können. Es steht zur Diskussion, ob die vorhandene *queryOrder (5.4)* Funktion dazu ausreicht.

Bestelloptionen
Die für eine Zentrale möglichen Bestelloptionen und die angebotenen Leistungen einer Zentrale („Taxi-Radar“, Preisermittlung) müssen dem Bestellsystem im Vorfeld einer Bestellung bekannt sein, da das Einfluss auf die Darstellung der Benutzeroberfläche haben kann.

Abfrage von Wagen ohne *queryOrder*
Es steht zur Diskussion, ob hier eine weitere Funktion nötig ist.

8 Historie dieses Dokuments

8.1 Initiale Version Mai 2012

Generelle Konzeption zur Diskussion.

8.2 Version vom November 2012

Konkretisierung, Umsetzung der eingebrachten Verbesserungen

- Festlegungen und Konkretisierung der Zeit/Datumsdarstellungen
- Groß/Kleinschreibung der Feldbezeichner geändert, gemäß Java Naming Conventions (siehe http://www.oracle.com/technetwork/java/codeconv-138413.html)
- Konkretisierung der Verfahrensweisen bei Telnet-like Transport, login
- Konkretere Definition der implementierten Funktionen

8.3 Version vom Februar 2013

Geringe Anpassungen und Korrekturen

9 Lizenzhinweise

Zum Zeitpunkt der ersten Veröffentlichung ist dieses Dokument als reiner Vorschlag, ohne irgendwelche verbindliche Inhalte oder verpflichtende Festlegungen zu verstehen. Es steht zur Diskussion und ist zum gegenwärtigen Zeitpunkt weder vollständig noch in allen Einzelheiten geprüft.

Das Dokument, **nicht** jedoch Implementationen der Schnittstellen auf Basis dieser Beschreibung, unterliegt der GNU GPL Version 2.

Allerdings sollten alle beteiligten Parteien zum Ziel haben, den Zugang zu Ihren jeweiligen Systemen gegenseitig zu ermöglichen um auf diese Weise die bestmögliche Lösung der Aufgabe (Abwicklung von Taxibestellungen) über Systemgrenzen hinweg zu erreichen.

10 Stichwortverzeichnis