Ride2Rail

FINAL SET OF REQUIREMENTS AND SPECIFICATION FOR COMPLEMENTARY TRAVEL EXPERT SERVICES

This project has received funding from the Shift2Rail Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement no. 881825

Project Acronym	RIDE2RAIL
Starting date	01/12/2019
Duration (in months)	30
Deliverable number	D2.6
Call Identifier	S2R-OC-IP4-01-2019
GRANT Agreement no	881825
Due date of the Deliverable	31/01/2021
Actual submission date	23/03/2021
Resposible/Author	Politecnico di Milano
Dissemination level	PU
Work package	WP2
Main editor	Politecnico di Milano
Reviewer(s)	Giuseppe Rizzi (UITP)
Status of document (draft/issued)	issued

Reviewed: yes

Contract No. 881825

Consortium of partners

PARTNER	COUNTRY
UNION INTERNATIONALE DES TRANSPORTS PUBLICS (UITP)	Belgium
FIT CONSULTING	Italy
OLTIS GROUP	Czech Republic
FSTECH	Italy
CEFRIEL	Italy
CERTH	Greece
EURNEX	Germany
EURECAT	Spain
POLIMI	Italy
UNIVERSITY OF NEWCASTLE UPON TYNE	United Kingdom
UNIFE	Belgium
UIC	France
UNIZA	Slovakia
ATTIKO METRO	Greece
INLECOM	Greece
FV-Helsinki	Finland
METROPOLIA	Finland

Contract No. 881825

DOCUMENT HISTORY		
Revision	Date	Description
0.8	15/1/2021	First revision, before gathering comments from CFM project members
0.9	11/2/2021	Incorporation of CFM project members feedback into document.
1.0	15/2/2021	Consolidated version, after comments from CEFRIEL and INLECOM, and after incorporation of modifications to Ride Tracking mechanisms discussed with OLITS and FST, ready for internal review.
1.1	16/02/2021	Review of the Document
1.2	19/03/2021	Final version for submission

REPORT CONTRIBUTORS		
Name	Beneficiary Short Name	Details of contribution
Politecnico di Milano	POLIMI	Main editor
Giuseppe Rizzi	UITP	Official Review

Disclaimer

Table of Contents

List of Figures

Contract No. 881825

List of Tables

1. EXECUTIVE SUMMARY

This deliverable presents the final set of requirements and the specifications of the modules to be developed within the RIDE2RAIL project. In addition, it analyses the software developed in previous Shift2Rail projects to determine whether the functions developed in RIDE2RAIL can be integrated into existing artifacts.

To determine the requirements and the specifications of the RIDE2RAIL modules, the deliverable follows a rather standard software engineering approach, and it first identifies the external actors with which the RIDE2RAIL modules will interact; then, it presents a set of use cases that provide an initial understanding of the functions to be realized in the project; from the use cases, a set of requirements are derived, then an initial model of the components to be developed in the project and their interfaces is provided in the form of a UML model.

The set of functions to be developed in the RIDE2RAIL projects can be grouped in two categories:

- those related to the interaction of passengers with the system, which should be provided through enhancements introduced in the Travel Companion;
- those related to the interaction of drivers with the system, which should be provided through a suitable user application (possibly itself an extension of the Travel Companion);
- and those related to the management of shared rides, which should be provided by a newly created Crowd-based Travel Service Provider.

The functions targeted to drivers include those for creating/deleting/modifying shared rides. The functions targeted to passengers, instead, include those aimed at presenting them offers that match well their preferences, but which also encourage a sustainable behaviour (e.g., take a shared ride vs. driving their own car when no public transportation is available to reach a train station or an airport). To provide better matches between passengers/drivers and offers, the RIDE2RAIL project will develop functions for keeping the preferences of passengers and drivers updated through automated learning mechanisms. Finally, the functions of Travel Service Providers offering shared rides include those for retrieving and booking shared rides, for issuing tokens suitable for their validation, and for monitoring their status.

The implementation of the functions developed in the RIDE2RAIL project should rely on (and be integrated into) existing platforms, and, in particular, those developed in previous Shift2Rail projects. The deliverable analyses the feasibility of the integration of the RIDE2RAIL modules into existing frameworks. To this end, it analyses the outcomes of three projects: (i) the Travel Companion developed within the ATTRACkTIVE Shift2Rail project, which can be the basis for the application offering the functions targeted to users; (ii) the CIGO! platform of the My-TRAC Shift2Rail project, which could be used to realize functions for the monitoring of shared rides; and (iii) the software components developed within the SocialCar H2020 project, which could be the basis for the implementation of the RIDE2RAIL Travel Service Provider. The outcome of this analysis is that the RIDE2RAIL project will aim to provide its user-oriented modules through an extension of the ATTRACkTIVE Travel Companion, and it will base the implementation of the Travel Service Provider for shared rides on the SocialCar infrastructure.

Contract No. 881825

2. ABBREVIATIONS AND ACRONYMS

ALM	Agreement Ledger Module
CFM	Call For Members
ETA	Expected Time of Arrival
OC	Open Call
R2R	RIDE2RAIL
R2Rapp	RIDE2RAIL application
R2RDrApp	RIDE2RAIL Driver Application
R2RCbTSP	RIDE2RAIL Crowd-based TSP
S2R	Shift2Rail
TC	Travel Companion
TSP	Travel Service Provider
TT	Trip Tracker
UC	Use Case

Contract No. 881825

3. BACKGROUND

The present document constitutes the Deliverable D2.6 “Final set of Requirements and Specification for complementary Travel Expert Services” in the framework of WP2 “Travel behavior and System Requirements”, and in particular Task T2.3 “Requirements and specifications for complementary travel expert services in the Shift2Rail IP4 ecosystem” of the RIDE2RAIL project (S2R-OC-IP4-01-2019).

It contributes as well to WP3 “Development of enhanced Travel Companion and Ride-sharing TSP” of the RIDE2RAIL project (S2R-OC-IP4-01-2019).

The present deliverable is the natural evolution of Deliverable D2.3, “First set of Requirements and Specification for complementary Travel Expert Services” [1], which was submitted at M8. More precisely, it revises and expands D2.3 based, on one hand, on the first developments and discussions that have been carried out in WP3 and, on the other hand, on the feedback received from CFM project members during scheduled Collaboration Meetings. The revisions and modifications introduced in D2.6 with respect to D2.3 are the following:

- Use Case UC7 has been revised to reflect feedback from CFM project members concerning what they see as the best way to integrate the functions developed by RIDE2RAIL within the S2R ecosystem.
- The specification has been completed with definitions related to the Agreement Ledger function, which was only hinted at in D2.3.
- In Chapter 8, which concerns the feasibility of integrating RIDE2RAIL functions into existing frameworks, and in particular in the S2R ecosystem, a section has been added to highlight possible problems concerning this integration, based on discussions and feedback received from CFM project members.

4. OBJECTIVES/AIM

The general goal of this deliverable is to provide the basis for the development of the software modules that will be the focus of WP3 "Development of enhanced Travel Companion and Ride-sharing TSP". To this end, it follows two directions.

First, it defines the first set of requirements and the specifications for the software modules that will be developed within the RIDE2RAIL project.

Second, it analyses the outcomes of several companion projects – especially in the context of the Shift2Rail initiative – to scout existing technologies and frameworks that can be the basis for the implementation of the RIDE2RAIL components, and it studies the feasibility of doing so.

Contract No. 881825

5. CONTENT OF DELIVERABLE

The deliverable is structured in three parts.

In the first part, the requirements for the RIDE2RAIL (R2R) system are presented. To this end, first a set of use cases are identified, which help define the main functions that the system will offer. From these use cases, a set of requirements that cover the use cases is derived.

In the second part, an initial specification of the R2R system is provided in terms of UML diagrams. The diagrams describe the main functions offered by the various elements of the system and the interactions among them. The diagrams also describe possible interactions of the R2R system with the existing S2R ecosystem.

In the third part, the feasibility of building the R2R components from existing ones is discussed. Also, the possibility to integrate the R2R components in the existing S2R ecosystem is investigated.

The contents of the document, and in particular the UML model that constitutes the specification of the R2R system, might be subject to further evolutions as the developments within WP3 progress. In particular, the UML model will be kept aligned with the actual design of the functions that is being carried out within WP3, and which has not yet been completed. In this regard, this document describes use cases (and corresponding requirements) that capture what have been identified as the best solutions to tackle needs of stakeholders. However, the actual implementation and demonstration of these solutions will heavily depend on the possibility of the existing S2R components to handle the necessary interactions identified in this document.

6. REQUIREMENTS ANALYSIS

This section defines the features that the R2R system will develop. It first identifies the scope of the system, that is, the macro-areas in which it contributes with innovative functions. Then, it describes the main concepts related to the R2R project. In particular, it identifies the actors involved in the system, separating them in two categories: external actors, who interact with the system by providing inputs or by providing functions used by the R2R system; and internal actors, who are the high-level elements of the R2R system. It also identifies the main high-level functions provided by the R2R system. The identified concepts are then used to define the use cases of the system, from which an initial set of requirements for the system is derived.

6.1. Scope of the system

The R2R system covers two related, but separate, aspects.

On one side, it introduces innovative functions targeted to travellers, which aim to steer them towards offers that are best suited to their preferences, but also that foster a virtuous, sustainable behaviour in the traveller (for example by incentivizing the use of modes of transportation that have low carbon footprint).

On the other side, it develops a stand-alone service, the so-called Crowd-based Travel Service Provider (R2RCbTSP), which allows private car owners to share their car rides with other travellers (see [2] for a definition of shared ride). From the point of view of the S2R ecosystem, the R2RCbTSP should behave as any other TSP, so it should offer functions that allow the system to retrieve travel episodes, book them, validate them, etc. Notice that the R2RCbTSP provides a front-end, the so-called R2R Driver Application (R2RDrApp), to private car owners to handle their rides.

To illustrate the role that the R2RCbTSP plays in the S2R ecosystem, a pair of scenarios are hereby introduced.

Scenario 1: Passenger using a shared ride: Jane is an engineer who lives in Crema, a town outside Milan that does not have a railway connection to Milan. Jane needs to go to Paris for a meeting. She would prefer to make the round trip in the same day, so she needs to leave very early in the morning. She opens her Travel Companion (TC) and puts in the data about her trip: she wants to leave on September 15th from her home in Crema, and she needs to reach Avenue Victor Hugo in Paris by 10.30. The TC returns a set of offers, which includes one in which Jane is supposed to join Chris, a fellow inhabitant of Crema, who has entered a car ride in the R2RCbTSP, which leaves from his home in Crema, but is also willing to stop in Crema's main square to pick other passengers up. Chris is going to Milan's central station to take a train to Rome, but on his way to central station he is willing to drop someone off at Linate airport. Both Chris and Jane have indicated that they prefer to travel with quiet people who do not like to talk much during the trip. Jane selects the offer that includes the shared ride offered by Chris. The system informs Chris of Jane's selection, and Chris accepts it, so the system books the trip for Jane.

Scenario 2: Traveller deciding to take their car for part of a trip, and then offering it as a shared ride. Chris lives in Crema and he wants to go to Rome to visit some friends in mid-September. Chris opens his TC to look for travel solutions, and inputs the information about his trip: he wants to leave on September 15th very early in the morning to have the full day to spend with his friends. The TC returns a set of offers for his trip, including one in which Chris is supposed to take his car to reach Milan's central station, then take a high-speed train to Rome. Since Crema is not well connected to Milan through public transportation, especially very early in the morning, Chris decides that taking his own car is the best option to reach Milan's central station, and he selects the corresponding offer. Since he chose to take his own car, Chris decides to offer the ride as a shared one and take on passengers along his way to Milan's central station. So, he opens the R2RDrApp and inputs the information about his ride to Milan's central station, including his willingness to stop along the path to pick up passengers, and the places where he is amenable to stop. The R2RDrApp sends the information to the R2RCbTSP, which includes the new ride among the available ones.

6.2. Description of concepts involved

6.2.1. Actors

The R2R system interacts with several types of actors, both humans and machines. We list them below and provide a brief explanation for each of them (see Figure 1 for an overview of the actors).

- **Passenger:** at the core of the S2R system (hence also of the R2R system) is the Passenger, who is the person who has a mobility need, and who actually performs the trip (the term comes from the glossary used in the MaaSive project, see also Section 6.2.3). In general, the trip taken by a Passenger might have been set up by an intermediary (e.g., a person from a travel agency). However, R2R focuses on actual Passengers (i.e., the people actually taking the trips, not intermediaries), whose preferences, for example, matter the most when planning a trip. Consequently, we assume that the Passenger is also the person shopping for offers, etc. A Passenger uses the TC to set up and execute a trip.
- **Driver:** the Driver is the other key actor in the R2R system. A Driver offers shared rides, which are performed through a personal vehicle, through the

R2RCbTSP. The ownership of the vehicle which is used to perform the shared ride is not relevant. However, a Driver must satisfy all legal constraints for driving a vehicle and comply with the R2R agreement requirements between the Driver and the Crowd-based TSP (as they will elaborated in Deliverable D3.5 – "Ride-Sharing Agreements Ledger Module"). In principle, a Driver might not have a mobility need, but they might simply offer a shared ride for other reasons (possibly economic).

- R2RPassenger: a R2RPassenger is a Passenger who takes (or has agreed to take) a shared ride offered by a Driver.
- TravellingDriver: when the Driver is also someone who has a mobility need, they become a TravellingDriver.
- Traveller: Driver and Passenger are separate types of actors (though a Passenger can also be a Driver, see the concept of TravellingDriver). However, it is sometimes useful to refer to either type of actor without distinguishing them. For this reason, we introduce the notion of Traveller, which generalizes both Passenger and Driver.
- GuestUser: a user of the TC who has not (yet) authenticated (or registered) themselves with the system. After a GuestUser signs up/in, they become a Passenger, or a Driver (or a TravellingDriver).
- S2R ecosystem: this is the collection of modules, services, applications developed within the S2R framework by projects other than R2R.
- IncentiveProvider: this[1] is an external actor (typically a service) that can provide incentives to steer the Passenger towards selecting certain offers instead of others. The R2R system does not offer incentives itself, but it is a conduit to the Passenger of incentives offered by an IncentiveProvider. An analysis of several possible types of incentives is presented in [3].

[1] From [3]: "The ***Incentive Provider (Entity)*** represents the entity proposing and responsible (also legally) for a given *Incentive*. The *Incentive Provider* may be a TSP or an organization, an association, or a government body. The *Incentive Provider* determines the *Incentive Mechanism* and the *Incentive Conditions* for the *Incentive*."

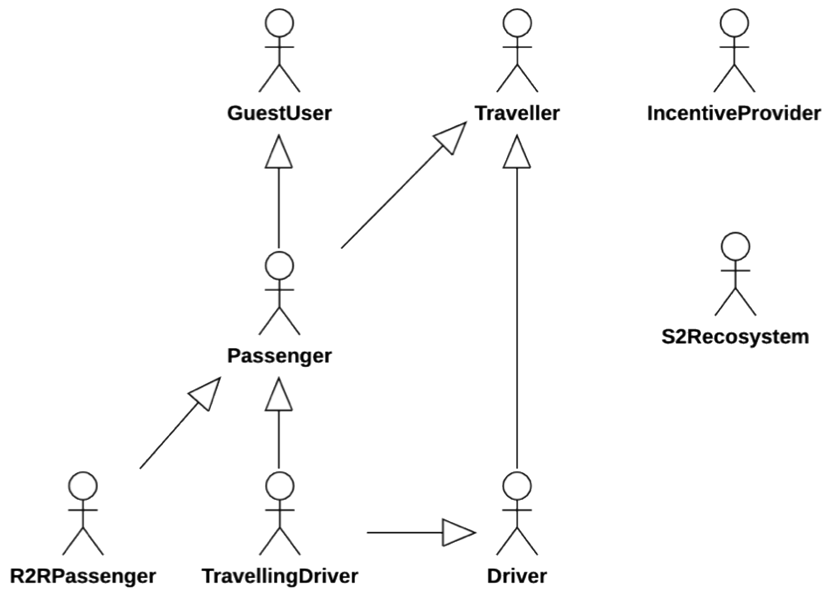

Figure 1 - Overview of the actors involved in the R2R system

6.2.2. High-level R2R system elements

At a high level, the R2R system includes the following main elements.

- **R2R application on the Passenger side (R2Rapp):** R2R will develop a number of features that are tailored to the experience of Passengers. Since Passengers interact with the S2R ecosystem through the TC, the features developed by R2R will have to be offered through a TC. Indeed, though it is not the goal of R2R to develop a full-fledged TC, R2R functions need to be part of a TC to be experienced by a Passenger (possibly a R2RPassenger). Ultimately, the resulting TC will be an *enhanced* one, which will include new features, in addition to the existing ones. In the rest of this document, we indicate through the term R2Rapp the enhanced TC that includes the Passenger-oriented features developed within the R2R project. Such enhanced TC should be an extension of an existing one that has already been developed within the S2R framework. Section 8 discusses the feasibility of extending existing TCs.

- **R2R application on the Driver side (R2RDrApp):** The R2R system needs to interact not only with R2RPassengers, but also with Drivers (e.g., to set up shared rides). This interaction will be achieved through a mobile application, which we indicate in this document as R2RDrApp. Since, in principle, a Driver does not need to also be a Passenger, the development of the R2RDrApp as an extension of an existing TC or as a separate application is a design decision that different implementations of the Sthat will be tackled later on.

- **R2R Crowd-based TSP (R2RCbTSP):** The R2RCbTSP is the element of the R2R system that handles the management (creation, storage, offering, execution, etc.) of shared rides and that interacts with the R2R Agreement Ledger Module managing the associated ride sharing agreements and smart contracts.

- **R2R Agreements Ledger Module (ALM):** The aim of this element is to provide functions that allow the secure storage of relevant data concerning shared rides coming from the R2R system, but also segments that are not shared rides and that come from the S2R ecosystem. In addition, the monitoring of events through the execution of smart contracts establishes a layer of trust between the actors of the whole ecosystem and enables the automatic resolution of any discrepancy between them.

6.2.3. Terminology used

The terminology used in this document is aligned with the glossary defined in the MaaSive project [4] (which has also been analysed in [3]), so we refer the reader to that document for an explanation of the meaning of the terms used.

6.2.4. List of high-level functions to be developed in RIDE2RAIL

At a high level, the R2R system will include the following main functions:

- **Itinerary offer categorization:** this function enriches existing offers with their category. For a discussion of offer categories in the context of the R2R project see [3].

- **Incentive mechanisms towards a more sustainable behaviour:** this function determines, for a given offer, the incentives that can be applied to it. A list of possible incentives has been defined in [3].

- **Learning of contextual user preferences:** this function deals with the problem of learning Passenger preferences given their activity on the TC. Having accurate user preferences is crucial to be able to steer the Passenger's choices towards the selection of the most suitable offers. However, the user cannot be expected to spend a lot of time accurately defining their preferences, so this function will relieve the user of this burden by automatically learning the preferences given the user behaviour.

- **Crowd-based TSP functions:** these are all functions related to the operation of the R2RCbTSP, and, in particular, the management of the shared rides: creation, offering, booking, validation, tracking, etc.

6.3. Use Cases

The use cases described in this section concern aspects that are specific to the R2R system with respect to other modules and functions that are available in the S2R ecosystem.

For example, it is clear that a Passenger would need to register to the system. However, this use case is already covered by TCs that have been developed under the S2R IP4 umbrella, and the R2R system does not seem to require specific actions concerning registration (at least from the Passenger's side). Therefore, a Passenger registration use case is not tackled in this document. However, let us notice that the UML model of Section 7 includes an extended model of preferences, which is part of the ongoing work of WP2, and which was finalized in deliverable D2.4 – "Final conceptualization of choice criteria and incentives".

Conversely, the registration of a Driver is specific to R2R, because the information needed from a Driver is different from that required from a Passenger, so the use case is covered in this document.

We tag each use case with a label, depending on which high-level function it covers:

- CAT: Itinerary offer categorization
- INC: Incentive mechanisms towards a more sustainable behaviour
- LUP: Learning of contextual user preferences
- CBTSP: Crowd-based TSP functions
- ALM: Agreement Ledger Module functions

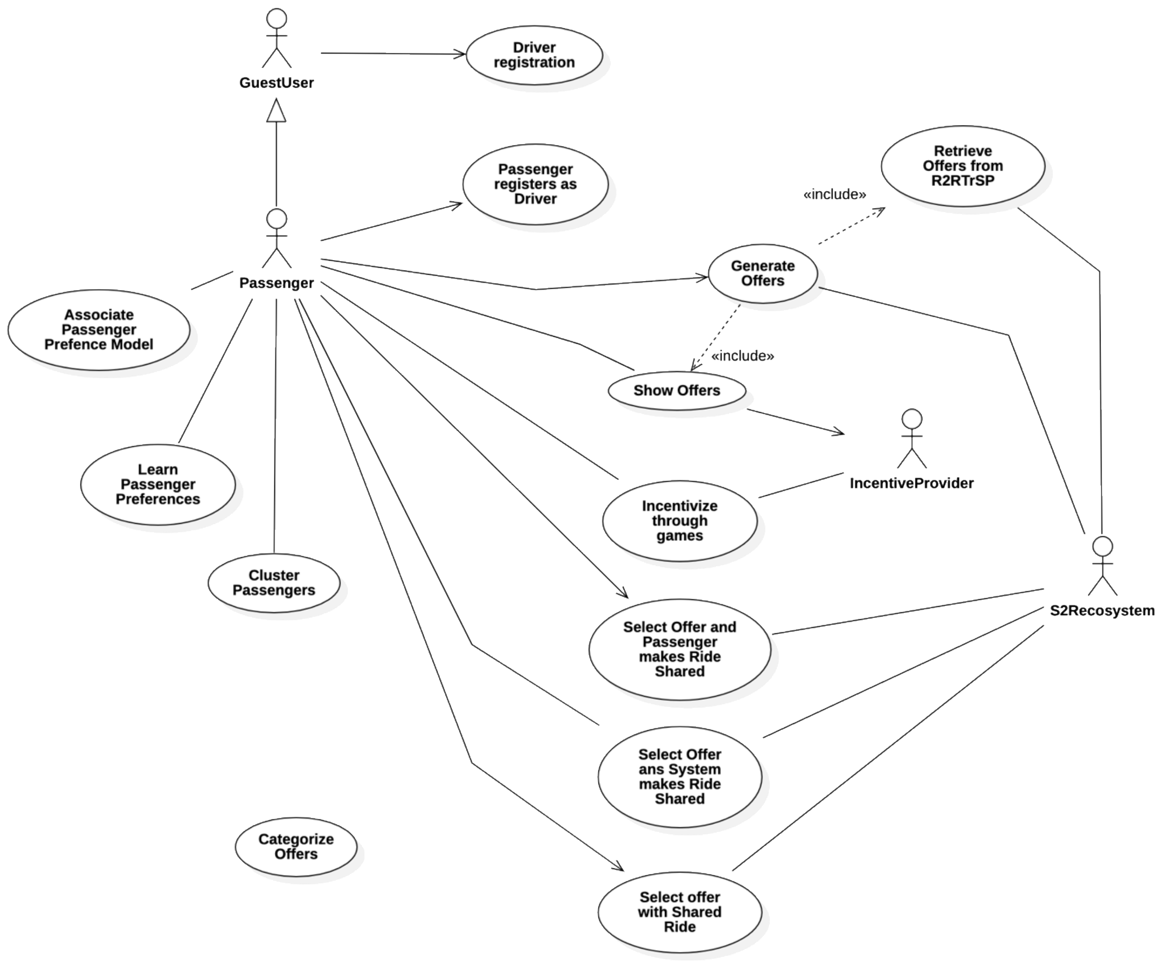
GuestUser
Driver registration
Passenger registers as Driver
Retrieve Offers from R2RTrSP
«include»
Generate Offers
Passenger
Associate Passenger Prefence Model
«include»
Show Offers
IncentiveProvider
Learn Passenger Preferences
Incentivize through games
S2Recosystem
Cluster Passengers
Select Offer and Passenger makes Ride Shared
Select Offer ans System makes Ride Shared
Categorize Offers
Select offer with Shared Ride

Figure 2 - Overview of the UCs involving the Passenger

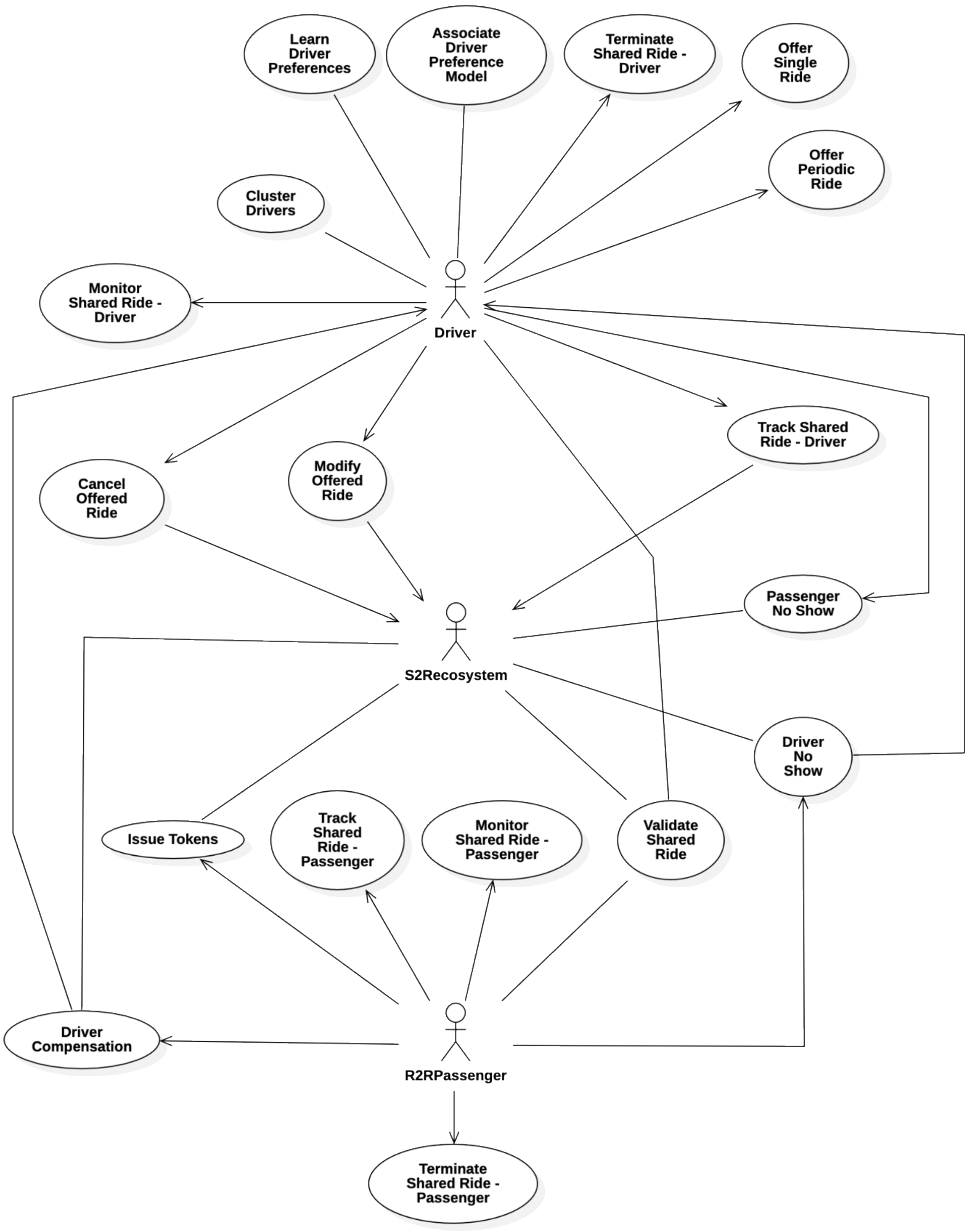

Figure 3 - Overview of the UCs involving the Driver and the R2RPassenger

6.3.1. User registration:

The user registration phase includes the use cases needed to feed the user profile system, where information about users of R2R are stored. Such information consists of personal data of both Driver and Passenger, including their preferences needed to manage the multimodal itinerary provision and the matchmaking features.

Some of these data may already be present in the TC and at the current status of design it is not clear which components will be used to manage each use case; in case of different components storing personal data (e.g., Passenger data stored in a different component than Driver data), it is required to design data flows and consider legal implications (GDPR and security issues).

We focus here on the most relevant use cases, especially from the point of view of information to be provided. Other, rather standard UCs are not included for brevity; for example, UCs for logging in, and for modifying user information are not listed here. Logging in the system is a standard operation that does not require specific treatment in the R2R context; similarly for the modification of data previously inserted.

Use Case UC1: Driver registration

Involved high-level functions.

CBTSP, LUP

Description

Table 1 Use case UC1: Driver registration

Actors	GuestUser (Driver), R2RDrApp, R2RCbTSP
Description	This use case illustrates the actions performed by a GuestUser when they decide to register as Driver in the R2R system.
Pre-conditions	GuestUser has installed R2RDrApp on personal device.
Story	1. GuestUser selects the “register as driver” option in the R2RDrApp. 2. GuestUser selects a TSP among the available ones that provide ride-sharing service. 3. They insert the necessary information: a. name b. surname c. date of birth d. username

	e. password f. address g. email h. phone number i. Information about the car (or cars) used: model, colour, year, license plate 4. R2RDrApp creates the profile and GuestUser becomes Driver. 5. R2RDrApp encourages Driver (by means of appropriate notifications) to fill optional fields to receive personalized services and the Driver is prompted to review and agree with the terms of their agreement with the Crowd-based TSP. 6. Driver approves providing optional fields. 7. R2RDrApp shows the optional fields. 8. Driver fills the optional fields. 9. R2RDrApp updates the Driver's personal data. 10. R2RDrApp informs R2RCbTSP of the new Driver. 11. R2RCbTSP updates the list of Drivers.
Alternatives	In Step 6, Driver clicks on Later. R2RDrApp schedules a reminder for them. In Step 6, Driver clicks on No. R2RDrApp terminates the rest of procedure.
Observations	Providing information such as the driving license number is probably not necessary if the Driver simply uses a private vehicle (it is the responsibility of the vehicle owner to guarantee that the Driver is allowed to drive it). Instead of through a mobile app (possibly included as part of a TC), GuestUser might register through a web app. The information (or some information) about the car could be retrieved from a public registry. Step 2 has been introduced to take into account the possibility that there might be more than one provider of ride-sharing services, and the user might want to select their preferred one. The actual registration might include additional, rather standard steps, for example to verify the existence of the email. As mentioned above, these are somewhat standard mechanisms that are not detailed here.

Extensions	• step 3a: the information of Driver is sent to the Learning Component to create a personal model for Driver and update the preference model(s).

Use case UC2: Incentivizing the newly registered Passenger to become Driver

Involved high-level functions.

CBTSP, INC, LUP

Description

Table 2 Use case UC2: Incentivizing the newly registered Passenger to become Driver

Actor	R2Rapp, Passenger (Driver), R2RCbTSP
Description	This use case illustrates the interaction between the newly registered Passenger and R2Rapp when R2Rapp incentivizes the Passenger to create their Driver profile.
Pre-conditions	During registration, Passenger indicated that they can use a car to perform a trip
Story	1. R2Rapp receives the registration of a Passenger. 2. R2Rapp notifies (by means of notification, message, e-mail, etc.) Passenger about the possibility of sharing rides and incentivizes them to fill the Driver profile. 3. Passenger logs into R2Rapp. 4. R2Rapp promotes the "Become a Driver" message with Yes, No and Later buttons. 5. Passenger clicks on Yes. 6. R2Rapp opens the Driver registration form. 7. Passenger inserts the necessary additional information: a. TSP to be used to offer shared rides b. Information about the car (or cars) used: model, colour, year, license plate 8. R2Rapp creates the profile and Passenger becomes Driver in this UC. 9. R2Rapp encourages Driver (by means of appropriate notifications) to fill optional fields to receive personalized services and and the Driver is

	prompted to review and agree with the terms of their agreement with the Crowd-based TSP. 10. Driver approves providing optional fields. 11. R2Rapp shows the optional fields. 12. Driver fills the optional fields. 13. R2Rapp updates the Driver's personal data. 14. R2Rapp informs R2RCbTSP of the new Driver. 15. R2RCbTSP updates the list of Drivers.
Alternatives	On step 4, Passenger chooses "No" and the procedure finishes. On step 4, Passenger chooses "Later", which causes R2Rapp to redo the procedure according to the predefined schedule.
Observations	Becoming a Driver is seen as part of the effort to incentivize a virtuous behaviour from Passenger, because it is the first step towards providing shared rides. Without registering as Driver, the Passenger cannot offer shared rides when the occasion arises, so this is a pre-emptive step in this direction. As in the case of Driver registration, inserting the driving license does not seem to be relevant, because it is the responsibility of the owner of the private vehicle to make sure that Driver has the right to drive it.
Extensions	• Steps 5a and 10a: the information of the Driver is sent to the Learning Component to create a personal model for Driver and update the preference model(s).

6.3.2. Offer ride

The next set of UCs is related to the operations that are performed to create and modify shared rides. As such, all UCs concerned with Drivers, rather than Passengers. We foresee two types of rides offered through the R2RCbTSP: single rides (executed once) and periodic ones.

All UCs refer to a single R2RCbTSP, because we assume that the Driver has selected beforehand what Crowd-based TSP they want to use.

Use case UC3: Offering a single ride

Involved high-level functions.

CBTSP, LUP

Description

Table 3 Use case UC3: Offering a single ride

Actors	Driver, R2RDrApp, R2RCbTSP
Description	This use case illustrates the actions performed by a driver when they decide to offer a new single ride (i.e., one that is executed only once, in a precise day) through the system for a maximum number of available seats and a maximum number of carried items.
Pre-conditions	Driver has logged into the system. Driver has planned to drive their car from address A to address B. The trip will take place on day DD, starting at time T at the starting point A where people pick up is possible/authorized.
Story	1. Driver selects the "offer ride" option in the R2RDrApp. 2. They select "single ride" as the type of ride. 3. They insert the parameters of the ride: a. Date (DD) b. expected start time (T) of ride, with some tolerance (t) c. duration d. starting point (A) e. possible intermediate stops (A1, A2, …) f. ending point (B) g. number of available seats h. cost of the ride i. vehicle j. number and size of allowed carried items (e.g., luggage) k. restrictions on items that can be carried (e.g., no guns) l. number, size and type of allowed pets m. PRM and Aid Tool compatibility n. preferred or allowed sex/Language/… o. smoking preference p. preferences related to types of passengers – chattiness – ok with keeping radio on

	– cultural interests – … 4. R2RDrApp informs R2RCbTSP of the new ride. 5. R2RCbTSP updates the list of rides it offers.
Alternatives	In step 3, instead of a single date, Driver might indicate a set of dates (still, the ride would not be periodic, but offered only on those precise dates, always at the same time). In step 3, Driver could indicate, for example through the help of R2RDrApp, the path that they will take, if they allow for intermediate stops. In step 3, Driver could indicate their willingness to make intermediate stops and/or make detours to accommodate passengers, with maximum tolerance for delays caused by stops/detour. Intermediate stops might be entered by the Driver themselves, or suggested by the system (probably the R2RCbTSP), based on its knowledge of the area in which the ride takes place. In step 3, Driver could indicate whether they want/need to approve of passenger joining the shared ride (or, they might automatically authorize any passenger). Driver could indicate the willingness to offer a roundtrip; if selected, the system should open a new partially filled-in form. The cost of the ride, instead of being input by the Driver, might be computed automatically by R2RCbTSP.
Observations	A and B are generic addresses, for example the home and office addresses of Driver, or a train station, or the stadium. The *duration* parameter in step 3 might be computed, instead of input by the user, to avoid cheating on times. Some of the information related to the ride might be optional; also, part of the information (e.g., preferences concerning types of passengers) might be retrieved from the Driver profile. The cost of the ride might be zero (the ride might be offered for free). We are considering the case where the Driver is not paid for the service they provide, but they receive incentives (e.g., discounts, see also UC12.

Contract No. 881825

Extensions	• Step 6: the parameters of the ride offered by Driver are sent to the Learning Component to update the preference model(s).

Use case UC4: Offering a periodic ride

Involved high-level functions.

CBTSP, LUP

Description

Table 4 Use case UC4: Offering a periodic ride

Actors	Driver, R2RDrApp, R2RCbTSP
Description	This use case illustrates the actions performed by a driver when they decide to offer a new periodic ride (i.e., one that is executed multiple times, with a same periodicity, in a certain range of days) through the system for a maximum number of available seats and a maximum number of carried items.
Pre-conditions	Driver has logged into the system. Driver has planned to drive their car from address A to address B. The trip will take place in days D1, D2, D3 of the week, each week from day SD, until day ED; each ride starts at time T at the starting point A where people pick up is authorized by the local urban/suburban authority.
Story	1. Driver selects the "offer ride" option in R2RDrApp. 2. They select "periodic ride" as the type of ride. 3. They insert the parameters of the ride: a. Days of the week in which the ride takes place (D1, D2, D3) b. Starting and ending day of the period of time in which the ride is available (SD, ED) c. expected start time (T) of ride, with some tolerance (t) d. duration e. starting point (A) f. intermediate Stops (A1, A2, …) g. ending point (B) h. cost of the ride i. number of available seats

	j. vehicle k. number and size of allowed carried items (e.g., luggage) l. restrictions on items that can be carried (e.g., no guns) m. number, size and type of allowed pets n. PRM and Aid Tool compatibility o. preferred or allowed sex/Language/... p. discount for specific communities/memberships q. smoking preference r. preferences related to types of passengers – chattiness – ok with keeping radio on – cultural interests – ... 4. R2RDrApp computes the set of single rides corresponding to the periodic ride entered by Driver. 5. R2RDrApp informs R2RCbTSP of the new set of rides. 6. R2RCbTSP updates the list of rides it offers.
Alternatives	In step 3, Driver could indicate, for example through the help of R2RDrApp, the path that they will take, if they allow for intermediate stops. In step 3, Driver could indicate their willingness to make intermediate stops and/or make detours to accommodate passengers, with maximum tolerance for delays caused by stops/detour. In step 3, Driver might be able to indicate some periods (or some days), within the range [SD,ED], in which the ride is not available. In step 3, Driver might be able to indicate some specific events (e.g., disruptions such as critical meteorological situations, significant traffic congestion caused by strikes) in which periodicity of the ride can suffer unpredicted modifications. In step 3, Driver could indicate whether they want/need to approve of passenger joining the shared ride. The cost of the ride, instead of being input by the Driver, might be computed automatically by R2RCbTSP.

Observations	See Use case UC3.
Extensions	• Step 7: the parameters of the ride offered by Driver are sent to the Learning Component to update the preference model(s).

Use case UC5: Cancel an offered ride

Involved high-level functions.

CBTSP, ALM

Description

Table 5 Use case UC5: Cancel an offered ride

Actors	Driver, R2RDrApp, R2RCbTSP, S2R ecosystem
Description	This use case illustrates the actions performed by a driver when they decide to cancel an offered ride.
Pre-conditions	Driver has logged into the system. Driver had planned to drive their car from address A to address B on day DD, starting at time T at the starting point A. Driver is no longer able to perform the ride.
Story	1. Driver selects the "cancel ride" option in the R2RDrapp. 2. They select the ride to be cancelled. 3. They insert the reason for cancelling. 4. R2RDrapp informs R2RCbTSP of the cancelled ride. 5. R2RCbTSP updates the list of rides it offers. 6. If the ride was already booked by some Passenger, notify S2R ecosystem (in particular, the Trip Tracking Orchestrator) of the cancellation and updates the Agreement Ledger.
Alternatives	
Observations	This UC focuses on a single ride. The single ride could be a single instance of a periodic ride. If the Driver needs to cancel a periodic ride, this could correspond to cancelling a set of single rides. If the cancelled ride was already booked by some R2RPassenger, they should have the possibility to rate the

	Driver. The reputation of Driver could be affected by the cancellation (possibly automatically). There should be a window of time when cancellation is allowed. This window of time should close some time before the start of the ride (it might be a configuration parameter of the R2RCbTSP). Any incentives that the Driver received for the ride should be taken back.
Extensions	• Step 3a: Driver's profile is updated with one cancellation action to update the reliability of Driver.

Use case UC6: Modify an offered ride

Involved high-level functions.

CBTSP, ALM

Description

Table 6 Use case UC6: Modify an offered ride

Actors	Driver, R2RDrApp, R2RCbTSP, S2R ecosystem
Description	This use case illustrates the actions performed by a driver when they decide to modify an offered ride.
Pre-conditions	Driver has logged into the system. Driver had planned to drive their car from address A to address B on day DD, starting at time T at the starting point A. Driver is no longer able to perform the ride as planned, and wants to modify it.
Story	1. Driver selects the "modify ride" option in the R2RDrApp. 2. They select the ride to be modified. 3. The Driver choses what parameters they need to change about the ride (ETA, duration, route etc.). 4. R2RDrApp informs R2RCbTSP of the modified ride. 5. R2RCbTSP updates the list of rides it offers. 6. If the ride is to be taken also by some Passenger, notify S2R ecosystem (and in particular the Trip Tracking Orchestrator) of the modification and updates the Agreement Ledger.

Contract No. 881825

Alternatives	Once the ride is created it can no longer be modified (or it can be modified only if no Passenger has booked it, yet) and the Driver has a limited amount of options available like signalling a necessary delay or cancellation.
Observations	A time lock can be set in place to disallow last minute changes to the ride unless the Driver wants to cancel it entirely. Modifications might be allowed also while the ride is ongoing. If the modified ride was already booked by some R2RPassenger, its modification might affect the R2RPassenger or not. The modification might result in the cancellation of a Travel Episode used by R2RPassenger, in which case R2RCbTSP should notify of the cancellation through the S2R ecosystem. In case the Travel Episode is cancelled, affected R2RPassengers should have the possibility to rate the Driver. The reputation of Driver could be affected by the modification (possibly automatically). If the incentives associated with the modified ride change as a consequence to the modification, they should be updated.
Extensions	• Step 3a: Driver's profile is updated with one modification action to update the reliability of Driver, if the modification affects some R2RPassenger.

6.3.3. Shop for offers that include shared rides

The UCs of this section focus on the operations to retrieve and show offers to Passengers. Retrieving offers is one of the basic functions offered by a TC, so many of the operations mentioned in the next UCs are already available in existing TCs, and do not need to be implemented from scratch by R2R. Still, they are included in the UCs to provide a context for the operations (e.g., ranking offers according to incentives and contextual user preferences) that are at the core of the R2R developments.

Use case UC7: Generating itinerary offers including possibly shared rides

Involved high-level functions.

CBTSP, LUP

Description

Table 7 Use case UC7: Generating itinerary offers including possibly shared rides

Actor	Passenger, S2R ecosystem, R2Rapp
Description	This use case illustrates the actions performed by Passenger when they look for an offer and the R2R system builds, through S2R ecosystem, offers that include shared rides.
Pre-conditions	Passenger has logged into the system. Passenger is looking for offers for going from address A to address B. The trip will take place on day DD, leaving at around time T.
Story	1. Passenger selects the "shop for trip" option in the R2Rapp. 2. They insert the options for the trip: a. date (DD) b. expected start time (T) c. starting point (A) d. ending point (B) e. indication of the type of the trip (e.g., work) f. number of people taking the trip (maximum three) g. number of carried items h. optional service preferences 3. R2Rapp sends a mobility request to S2R ecosystem. 4. S2R ecosystem returns with a set of itinerary offers, which include shared rides, ranked according to the user contextual preferences and to the incentives. 5. R2Rapp shows ranked offers to the Passenger.
Alternatives	In Step 2, Passenger could indicate that they have some mobility limitation, for example that they walk with the aid of crutches, or that they use a motorized wheelchair. In Step 2, Passenger could indicate that they are not interested in joining a shared ride that will perform intermediate stops. The S2R ecosystem might return a set of Itinerary Offers that does not include shared rides. The use case would be the same, the only difference being that in step 4 the set of offers would be different.

Contract No. 881825

Observations	Steps 1 and 2 should already be available in existing TCs, and they should not require a modification in the R2R project. They are listed here to provide the context of the parts that are more specific of R2R (e.g., the ranking of the offers). Steps 3, 4, 5 are further detailed in Use cases UC7a and UC7b.
Extensions	• Step 3a: the parameters of the mobility request by Passenger are sent to the Learning Component to update the preference model(s).

Use case UC7a: Generation of itinerary offers through the S2R ecosystem

Involved high-level functions.

CBTSP

Description

Table 8 Use case UC7a: Generation of itinerary offers through the S2R ecosystem

Actor	S2R ecosystem, R2Rapp, R2RCbTSP, IncentiveProvider
Description	This use case illustrates the interaction between R2Rapp, S2R ecosystem, and R2RCbTSP when offers are built that include shared rides.
Pre-conditions	R2Rapp has received a mobility request from a Passenger. The mobility request is for offers for going from address A to address B. The trip will take place on day DD, leaving at around time T. Part of the trip can be covered through shared rides offered by R2RCbTSP. In particular, R2RCbTSP has many rides, each one with its own separate starting point A_i and ending point B_i, that can cover part of the trip from A to B. S2R ecosystem knows the contextual preferences that have been used to create offers O. Passenger has authorized the incentive mechanisms. Passenger has authorized R2Rapp to process the personal data.
Story	1. R2Rapp sends mobility request to S2R ecosystem.

	2. S2R ecosystem uses Interoperability Framework to find R2RCbTSP that can cover part of the request from A to B. 3. S2R ecosystem asks R2RCbTSP to provide Travel Episodes covering trip A⇢B. 4. R2RCbTSP returns a (possibly empty) set of shared rides covering (part of) trip A⇢B. 5. S2R ecosystem retrieves set of rides A_i⇢B_i from R2RCbTSP. 6. S2R ecosystem builds a set of offers O, among which there are offers O_i, each including a shared ride A_i⇢B_i among its travel episodes. 7. S2R ecosystem filters (possibly relying on the features offered by TSPs) the travel offers according to the limitations indicated by Passenger (if any). 8. S2R ecosystem categorizes each offer that belongs to set O. 9. S2R ecosystem interacts with IncentiveProvider to associate each offer with a set of available incentives (e.g., Eco Points to be converted in giveaways, discount of ride sharing payment). 10. S2R ecosystem ranks offers that belong to set O, according to contextual user preferences, to the compatibility of Passenger with drivers D1 and D2, and to incentives. 11. S2R ecosystem returns set O to R2Rapp.
Alternatives	R2RCbTSP does not directly offer any ride that covers part of the trip from A to B; however, it offers a “flexible” ride that can be rearranged (for example by adding an intermediate stop) to cover part of the trip from A to B. The ride could be rearranged in several ways; an intermediate stop could be added along the planned route; or a stop could be introduced that requires a detour, but within the acceptable limits stated by the driver. If a rearranged ride also involves other passengers, they must be informed and they must be given the possibility to cancel their segment. Alternative pre-condition: Passenger did not authorize the incentive mechanisms; in this case, in Step 4 the ranking will be done using only the contextual preferences. Alternative pre-condition: Passenger did not authorize R2Rapp to process their personal data; then, in Step 4

	R2Rapp computes the scores for the travel offers using default settings (e.g., the faster the trip, the higher the score).
Observations	By “shared ride $A_i \dashrightarrow B_i$” we mean a trip from A_i to B_i, that occurs at a certain time, in a certain date. This UC is introduced to highlight the interaction with existing elements of the S2R ecosystem. As such, it spans a number of steps that are not in the purview of the R2R system (e.g., Step 1, Step 4, which should already be implemented by existing S2R ecosystem elements).
Extensions	

Use case UC7b: Presentation of itinerary offers to Passenger

Involved high-level functions.

LUP, CAT, INC

Description

Table 9 Use case UC7b: Presentation of itinerary offers to Passenger

Actor	Passenger, R2Rapp
Description	This use case illustrates the interaction between Passenger and R2Rapp after itinerary offers are retrieved from S2R ecosystem.
Pre-conditions	The R2Rapp has received a set of itinerary offers O from S2R ecosystem. Among the offers there is a set of offers O_i, each including a shared ride $A_i \dashrightarrow B_i$ (offered by Driver D_i). R2Rapp knows the contextual preferences that have been used to create offers O. Passenger has authorized the incentive mechanisms. Passenger has authorized R2Rapp to process the personal data.
Story	1. R2Rapp shows ranked offers to Passenger. 2. While Passenger peruses the ranked offers, R2Rapp monitors their behaviour to learn about their preferences and provides incentives to select more virtuous offers

Alternatives	After receiving the ranked offers (Step 1), Passenger may decide to re-sort the provided list according to the possible categories (e.g., from the fastest to the slowest). After receiving the ranked offers (Step 1), Passenger may decide to apply filter(s) on the provided list using the R2Rapp filter options (e.g., Price Range, Segments, Duration, Vehicle, etc.).
Observations	See Use case UC7a. We need to make clear for the traveller the advantages of opting for the journeys that take into consideration ride sharing. For example, the system could point out reductions in travel time, or eco-friendly advantages, in terms of CO_2 Reduction, that are enabled by the ridesharing opportunity.
Extensions	• Step 2a: the activities collected from Passenger's interactions with the presented ranked offers (e.g., ignoring, applying further filter options, applying sort functions) are sent to the Learning Component to update the preference model(s). Collection of the activities is dependent on the availability of specific functions, and on the respect of the GDPR. The minimum required activities to be collected include the set of ranked offers which have been showed to the user and the booked one.

6.3.4. Incentivizing virtuous behaviours

One of the goals of the R2R project is to incentivize a more sustainable (virtuous) behaviour on the part of Passengers. Previous UCs (e.g., UC7b) focused on passive mechanisms based on showing first offers that are more virtuous. The next UC suggests a more complex mechanism that could be considered within the R2R project; however, the realization of such a UC is not a priority of the R2R project.

Use case UC8: Incentivizing Passenger through Gamification mechanism

Involved high-level functions.

CBTSP, INC

Description

Table 10 Use case UC8: Incentivizing Passenger through Gamification mechanism

Actor	R2Rapp, Passenger, IncentiveProvider
Description	This use case illustrates the interaction between Passenger, IncentiveProvider and R2Rapp when a gamification mechanism is activated by IncentiveProvider to incentivize Eco-friendly behaviour. The goal of the gamification mechanism is to promote travel offers with lower CO_2 emissions. Target Users: Travellers related to region X.
Pre-conditions	Passenger has authorized incentive mechanisms. IncentiveProvider has detailed and published the Eco-friendly challenge.
Story	1. IncentiveProvider publishes the information of the game/challenge. 2. R2Rapp retrieves from IncentiveProvider the details of the game/challenge. 3. R2Rapp initializes the game/challenge according to the information received from IncentiveProvider. 4. R2Rapp retrieves the potential target Travellers (related to region X). 5. R2Rapp notifies target Travellers about the game/challenge and sends out the corresponding instructions. 6. Travellers join the game. 7. R2Rapp keeps track of Travellers' activity related to the game/challenge for those who agreed to participate in it. 8. When the game/challenge ends, R2Rapp sends the result to IncentiveProvider. 9. IncentiveProvider determines the list of winners and their prizes (if any) and notifies it to R2Rapp. 10. R2Rapp notifies winning Travellers about their prize.
Alternatives	
Observations	This use case can be modified also for Driver with a different game use case.
Extensions	

6.3.5. Select and book offers

The next UCs concern the steps undertaken after offers are presented to a Passenger and, more precisely, to the selection and booking of an offer. In this phase, if the offer includes a Travel Episode that is supposed to be taken through a private vehicle, the system might suggest making it a shared ride for sustainability reasons. We present two alternative processes for making a private ride a shared one: one in which the decision is made by Passenger at selection time, and one in which the R2R system flags an itinerary offer item as "shareable".

"Payment" for the ride is an issue that requires to be handled with some care. For legal reasons, it is not advisable that the Crowd-based TSP accept payments on behalf of the Driver. However, a Driver would expect to be compensated for the fact that they offer a ride to other people. Hence, instead of defining a use case for the "payment" of a shared ride, we define a UC concerning the "compensation" related to a shared ride, leaving open the possibility of interpreting the notion of "compensation" in different ways (e.g., discounts, bonus points, etc.).

Use case UC9: Selecting itinerary offer that includes a ride that becomes shared (Passenger's decision)

Involved high-level functions.

CBTSP, LUP

Description

Table 11 Use case UC9: Selecting itinerary offer that includes a ride that becomes shared (Passengers decision)

Actor	Passenger (TravellingDriver), S2R ecosystem, R2Rapp, R2RCbTSP
Description	This use case illustrates the interaction that occurs between Passenger, R2Rapp, S2R ecosystem and R2RCbTSP when Passenger selects for booking an itinerary offer that includes a car ride (where the driver is Passenger), which the Passenger decides to offer as a shared ride.
Pre-conditions	R2Rapp has shown to Passenger a list of itinerary offers, among which there is one that includes a car ride in which Passenger is supposed to take their own car. Passenger has already registered as Driver.

Contract No. 881825

Story	1. Passenger selects itinerary offer O_i, which includes a car ride $A_i \dashrightarrow B_i$ in which Passenger is supposed to take their own car. 2. R2Rapp asks Passenger whether they want to offer ride $A_i \dashrightarrow B_i$ as a shared one through the R2RCbTSP. 3. Passenger ticks the option of making $A_i \dashrightarrow B_i$ a shared ride (hence becomes a TravellingDriver). 4. R2Rapp automatically retrieves some parameters of the ride: a. Date (DD) b. expected start time (T) of ride c. duration d. starting point (A) e. ending point (B) 5. TravellingDriver revises the automatically retrieved parameters of the ride, possibly completes them, and insert missing ones: a. tolerance (t) of expected start time b. intermediate Stops (A1, A2, …) c. number of available seats d. vehicle e. payment mode f. number and size of allowed carried items (e.g., luggage) g. Restrictions on items that can be carried (e.g., no guns) h. number, size and type of allowed pets i. PRM and Aid Tool compatibility j. preferred or allowed sex/Language/… k. discount for specific communities/memberships l. smoking preference m. preferences related to types of passengers – chattiness – ok with keeping radio on – cultural interests – … 6. R2Rapp informs R2RCbTSP of the new ride. 7. R2RCbTSP updates the list of rides it offers.

Contract No. 881825

	8. R2Rapp sends offer O_i to S2R ecosystem for booking (except the itinerary offer item for the newly-created shared ride).
Alternatives	The Passenger is not registered as Driver. In this case, R2Rapp might consider UC2 before proceeding with Step 4.
Observations	See Use case UC3. See Use case UC7a. From the functional point of view, R2Rapp in this use case also performs operations that R2RDrApp performs in UC3. It is a design decision to make R2RDrApp a separate application from R2Rapp, or an internal module of R2Rapp
Extensions	• Step 1a: The list of presented offers along with their rank and distinguishing the chosen one (O_i) is sent to the Learning Component to update/correct the scoring. • Step 3a: The Passenger's preference to share ride is sent to the Learning Component to update the preference model(s). • Step 5a: The corrected/added parameters are sent to the Learning Component to update the preference model(s).

Use case UC10: Selecting itinerary offer that includes a ride that becomes shared (system's flag)

Involved high-level functions.

CBTSP, LUP

Description

Table 12 Use case UC10: Selecting itinerary offer that includes a ride that becomes shared (system's flag)

Actor	Passenger (TravellingDriver), S2R ecosystem, R2Rapp, R2RCbTSP
Description	This use case illustrates the interaction that occurs between Passenger, R2Rapp, S2R ecosystem and R2RCbTSP when Passenger selects for booking an itinerary offer that includes a car ride (where the driver is Passenger), which the Passenger decides to offer as a shared ride. Unlike UC9,

	in this case the itinerary offer item has pre-emptively been marked by the S2R system as “shareable” thanks to the fact that Passenger has signalled their willingness to share possible rides at the moment of sending the mobility request.
Pre-conditions	R2Rapp has shown to Passenger a list of itinerary offers, among which there is one that includes a car ride in which Passenger is supposed to take their own car, and which the system has suggested to make a shared one. When submitting the mobility request, Passenger has indicated that, if a returned itinerary offer includes a ride with a private car, they allow (or even prefer) for such a ride to be converted to a shared one.
Story	1. Passenger selects itinerary offer O_i, which includes a car ride $A_i \dashrightarrow B_i$ in which Passenger is supposed to take their own car, and which the system has flagged as “sharable” (hence Passenger becomes a TravellingDriver). 2. R2Rapp automatically retrieves some parameters of the ride: a. Date (DD) b. expected start time (T) of ride c. duration d. starting point (A) e. ending point (B) 3. TravellingDriver revises the automatically retrieved parameters of the ride, possibly completes them, and insert missing ones: a. tolerance (t) of the expected start time b. intermediate Stops (A1, A2, …) c. number of available seats d. vehicle e. payment mode f. number and size of allowed carried items (e.g., luggage) g. restrictions on items that can be carried (e.g., no guns) h. number, size and type of allowed pets i. PRM and Aid Tool compatibility j. preferred or allowed sex/Language/…

	k. discount for specific communities/memberships l. smoking preference m. preferences related to types of passengers – chattiness – ok with keeping radio on – cultural interests – … 4. R2Rapp informs R2RCbTSP of the new ride. 5. R2RCbTSP updates the list of rides it offers. 6. R2Rapp sends offer O_i to S2R ecosystem for booking (except the itinerary offer item for the newly-created shared ride).
Alternatives	
Observations	See Use case UC3. See Use case UC7a. See Use case UC9.
Extensions	• Step 1a: the list of presented offers along with their rank and distinguishing the chosen one (O_i) is sent to the Learning Component to update/correct the scoring. • Step 3a: the corrected/added parameters is sent to the Learning Component to update the preference model(s).

Use case UC11: Selecting itinerary offer that includes a shared ride

Involved high-level functions.

CBTSP, LUP, ALM

Description

Table 13 Use case UC11: Selecting itinerary offer that includes a shared ride

Actor	Passenger (R2RPassenger), S2R ecosystem, R2Rapp, R2RCbTSP, Driver, R2RDrApp
Description	This use case illustrates the interaction that occurs between Passenger, R2Rapp, S2R ecosystem, R2RCbTSP and Driver

	when Passenger selects for booking an itinerary offer that includes a shared ride offered by Driver.
Pre-conditions	R2Rapp has shown to Passenger a list of itinerary offers, among which there is one that includes a shared ride offered by Driver. R2RPassenger has authorized R2Rapp to share information about themselves with R2RCbTSP.
Story	1. Passenger selects itinerary offer O_i, which includes shared ride $A_i \dashrightarrow B_i$ offered by R2RCbTSP and handled by Driver (hence Passenger becomes R2RPassenger). 2. Passenger provides specific information needed by Driver, if not already included in mobility request (number of persons for the trip, number of carried items). 3. R2Rapp notifies S2R ecosystem that Passenger requests the booking of offer O_i. 4. S2R ecosystem books each itinerary offer item with its respective TSP, including the one corresponding to shared ride $A_i \dashrightarrow B_i$. To book the shared ride $A_i \dashrightarrow B_i$, S2R ecosystem contacts the R2RCbTSP. 5. R2RCbTSP notifies Driver (through R2Rapp) of the request to book shared ride $A_i \dashrightarrow B_i$ by R2RPassenger. The notification includes basic information about R2RPassenger concerning name, surname, email, phone number, number of carried items and number of persons for the trip. 6. Driver connects to R2RDrApp and sees the details of the requested ride and of R2RPassenger. 7. Driver accepts booking through their R2RDrApp. 8. R2RDrApp notifies R2RCbTSP that Driver accepted the booking. 9. R2RCbTSP updates the Agreement Ledger. 10. R2RCbTSP confirms to S2R ecosystem that the shared ride $A_i \dashrightarrow B_i$ has been confirmed. 11. S2R ecosystem confirms booking to Passenger's R2Rapp 12. R2RPassenger activates trip tracking through R2Rapp, and R2Rapp triggers the TT.

Alternatives	Driver refuses the booking of the shared ride. Driver provides an explanation for refusing the booking of the shared ride. Driver does not respond within a pre-defined delay, which amounts to refusing the booking (without providing an explanation). Authorizations are automatically granted depending on the preferences of R2RPassenger and Driver.
Observations	See Use case UC7a. The rules and protocols with which R2Rapp will handle Passengers' personal information should have already been defined within the S2R TC which will be the basis for R2Rapp (see also Section 8). Not only Driver, but also other passengers of the same ride might be notified (by R2RCbTSP) that somebody else is joining. R2RCbTSP keeps track of the behaviour of Driver for what concerns acceptance/rejection of rides. Rejection of rides could affect the reputation of Driver. A booking (in particular the ride-sharing part) could undergo a process of re-planning of the itinerary in case specific contingencies affect the original planning (e.g., traffic congestion, strikes, traffic disruptions, R2RPassenger no show). We need to consider the case in which R2RPassenger can cancel the booked ride up to a certain date. We need to consider specific communication provided by Driver concerning the geo localization of the pick-up place where the traveller will join the service. We need to consider specific penalties for R2RPassenger that books the service and does not join the ride by the stated time (R2RPassenger NO SHOW). We need to consider specific penalties for Driver that does not show up within a certain delay from the agreed pick-up time (Driver NO SHOW). We need to consider the opportunity to give the traveller specific indications about Driver and the vehicle used for the shared ride to assure them of the quality of the service.
Extensions	• Step 1a: the list of presented offers along with their rank and distinguishing the chosen one (O_i) is sent to the Learning Component to update/correct the scoring.

Use case UC12: Compensation of Driver for offering booked shared ride

Involved high-level functions.

CBTSP, ALM

Description

Table 14 Use case UC12: Compensation of Driver for offering booked shared ride

Actor	R2RPassenger, S2R ecosystem, R2Rapp, R2RCbTSP, Driver
Description	This use case illustrates part of the issuing process for a shared ride, that is, the interaction that occurs between R2RPassenger, R2Rapp, S2R ecosystem, R2RCbTSP and Driver after an itinerary offer that includes a shared ride $A_i \dashrightarrow B_i$ offered by Driver is booked and needs to be paid. The UC focuses on the part that is specific to the Crowd-based TSP. Payment of offer items that are not related to a shared ride occurs as prescribed by the S2R ecosystem.
Pre-conditions	Booking of itinerary offer that includes a shared ride offered by Driver has been successful.
Story	1. R2RPassenger initiates the payment process, providing the necessary information. 2. R2Rapp sends received information to the S2R ecosystem. 3. S2R ecosystem uses the information to notify R2RCbTSP of the selection of the ride by R2RPassenger. 4. R2RCbTSP computes the compensation for the Driver and updates their profile accordingly. 5. R2RCbTSP confirms finalization of shared ride to S2R ecosystem. 6. R2RCbTSP notifies Driver of the received compensation. 7. Finalization is confirmed to R2RPassenger. 8. R2RCbTSP updates the Agreement Ledger.
Alternatives	
Observations	Steps 1 and 2 are not further detailed, because they occur according to the mechanisms defined by the S2R ecosystem, independent of R2R.

Extensions	

Use case UC13: Generating tokens for issued itinerary offer that includes a shared ride

Involved high-level functions.

CBTSP, ALM

Description

Table 15 Use case UC13: Generating tokens for issued itinerary offer that includes a shared ride

Actor	S2R ecosystem, R2Rapp, R2RCbTSP, R2RDrApp
Description	This use case illustrates the interaction that occurs between R2Rapp, S2R ecosystem and R2RCbTSP after an itinerary offer that includes a shared ride offered by Driver is issued (i.e., it is paid for), so the tokens allowing Passenger to actually take the ride need to be generated.
Pre-conditions	Payment by R2RPassenger of itinerary offer that includes a shared ride offered by Driver has been successful.
Story	1. R2Rapp (of R2RPassenger) contacts S2R ecosystem to retrieve tokens to be used for the validation of the trip, including the shared ride. 2. S2R ecosystem contacts R2RCbTSP to retrieve token for shared ride. 3. R2RCbTSP generates the token and returns it to S2R ecosystem. 4. R2RCbTSP sends token to R2Rapp of Driver, so they can perform the validation when the shared ride starts. 5. R2RDrApp of Driver internally stores validation token. 6. S2R ecosystem returns token to R2Rapp. 7. R2Rapp stores token internally, ready to be used for the validation. 8. R2RCbTSP updates the Agreement Ledger.
Alternatives	
Observations	

Extensions	

6.3.6. Execute a trip that includes a shared ride

The next UCs focus on the phases of the execution of a shared ride: the initiation of the ride (and in particular the validation step), the monitoring of the execution (to detect any delays or problems that might arise), and the termination of the ride (with any related feedback).

Use case UC14: Validate part of trip performed through shared ride

Involved high-level functions.

CBTSP, ALM

Description

Table 16 Use case UC14: Validate part of trip performed through shared ride

Actor	R2RPassenger, R2Rapp, R2RCbTSP, Driver, R2RDrApp
Description	This use case illustrates the interaction that occurs between R2RPassenger, R2Rapp, R2RCbTSP, Driver and R2RDrApp when R2RPassenger executes the part of the trip carried out through a shared ride offered by Driver.
Pre-conditions	R2RPassenger has booked (and paid) an itinerary offer that includes a shared ride offered by Driver (i.e., the offer has been issued). Passenger is executing the trip, and they have reached the point of executing the shared ride. Both the Driver and R2RPassenger have logged into the R2RDrApp and R2Rapp, respectively
Story	1. The R2RPassenger boards the vehicle 2. The Driver selects the correct ride from their list in their R2RDrApp and so does the R2RPassenger in their R2Rapp. 3. The R2Rapp on the R2RPassenger's device shows the unique identification mechanism associated with them. 4. The Driver uses the R2RDrApp to perform the validation of the identification the Passenger is providing.

	5. A validation control message is sent to the R2RCbTSP that performs the needed checks. 6. If the validation is successful both the Driver and the R2RPassenger are notified. 7. The R2RCbTSP registers the presence of both parties on the vehicle and the trip can begin. 8. R2RCbTSP updates the Agreement Ledger.
Alternatives	The identification mechanism could be activated manually by letting the Driver check a document the R2RPassenger provides if this document number has been subscribed to the ride
Observations	Validation could be done automatically using GPS data, therefore requiring onboarding preferences for sharing location with R2Rapp. This process can be made bidirectional to provide means to the Passenger to verify they are indeed on the correct vehicle.
Extensions	

Use case UC15: Monitor execution of shared ride

This Use case is split in 3 parts: the Passenger's side, the Driver's side, and the R2RCbTSP side. The R2RCbTSP side is itself divided in 2 parts: one that concerns reactions to issues related to Driver, and one that concerns reactions to issues related to Passenger.

Use case UC15a: Monitor execution of shared ride (Driver's side)

Involved high-level functions.

CBTSP, ALM

Description

Table 17 Use case UC15a: Monitor execution of shared ride (Driver side)

Actor	Driver, R2RDrApp, R2RCbTSP
Description	This use case illustrates the interaction that occurs between Driver, R2RDrApp, and R2RCbTSP during the execution of a shared ride offered by Driver.
Pre-conditions	Driver has offered a ride that is taken by some passenger.

Story	1. Driver boards the car. 2. Driver selects the correct ride from their list in their R2RDrApp. 3. R2RDrApp on Driver's device starts sending the position of Driver to R2RCbTSP. 4. R2RDrApp on Driver's device periodically sends the position of Driver to R2RCbTSP. 5. Driver uses R2RDrApp to declare that shared ride has ended. 6. R2RDrApp stops sending the position of Driver to R2RCbTSP. 7. R2RCbTSP updates the Agreement Ledger with the specific events form the trip tracker (e.g. Delay).
Alternatives	R2RCbTSP might determine, based on the time and the date, when the ride starts and ends. R2RDrApp might periodically check with R2RCbTSP when the ride is supposed to start, and start sending the position automatically when the time approaches. Driver might have a menu to send custom alerts (e.g., "stuck in traffic") that might be then relayed to the system (and then to R2RPassenger).
Observations	
Extensions	

Use case UC15b: Monitor execution of shared ride (Passenger's side)

Involved high-level functions.

CBTSP

Description

Table 18 Use case UC15b: Monitor execution of shared ride (Passenger side)

Actor	R2RPassenger, R2Rapp, R2RCbTSP
Description	This use case illustrates the interaction that occurs between R2RPassenger, R2Rapp, and R2RCbTSP during the execution of a shared ride taken by Passenger.
Pre-conditions	R2RPassenger has booked (and paid) an itinerary offer that includes a shared ride offered by some Driver.

Story	1. R2RPassenger enables tracking. 2. R2Rapp on R2RPassenger's device starts sending the position of Passenger to R2RCbTSP. 3. R2Rapp on R2RPassenger's device periodically sends the position of Passenger to R2RCbTSP. 4. R2RPassenger uses R2Rapp to declare that their part of the shared ride has ended. 5. R2Rapp stops sending the position of R2RPassenger to R2RCbTSP.
Alternatives	R2RCbTSP might determine, based on the time and the date, when the ride starts and ends. R2Rapp might periodically check with R2RCbTSP when the ride is supposed to start, and start sending the position automatically when the time approaches. Passenger might have a menu to send custom alerts (e.g., "being late") that might be then relayed to the system (and then to the Driver).
Observations	R2RPassenger is being tracked before the start of the ride to potentially inform Driver of any delay of Passenger. This would be a peculiarity of a ride-sharing TSP, where not only the vehicles, but also passengers are tracked; the reason behind this lies in the peculiarity of the relationship between Driver and R2RPassenger in a shared ride, which is much more personal than in for a usual transport service. The actual realization of this UC is not prioritary, and might be left for future work.
Extensions	

Use case UC15c: Monitor execution of shared ride by TSP (Driver's side)

Involved high-level functions.

CBTSP, ALM

Description

Table 19 Use case UC15c: Monitor execution of shared ride by TSP (Driver's side)

Actor	R2RDrApp, R2RCbTSP, S2R ecosystem
Description	This use case illustrates the interaction that occurs between R2RDrApp, R2RCbTSP, R2Rapp and S2R ecosystem during

	the execution of a shared ride taken by Passenger and offered by Driver, to generate notifications about potential issues with the shared ride (e.g., delays) that are related to Driver.
Pre-conditions	Passenger has booked (and paid) an itinerary offer that includes a shared ride offered by a Driver. Tracking of shared ride has been initiated on Driver's side.
Story	1. R2RCbTSP periodically receives position of Driver from R2RDrApp. 2. R2RCbTSP compares current position of Driver with expected one. 3. If R2RCbTSP notices a discrepancy between Driver's expected and actual positions, it signals a problem to the S2R ecosystem. 4. S2R ecosystem (through R2Rapp) informs R2RPassenger that there is an issue with the shared ride. 5. R2RCbTSP updates the Agreement Ledger.
Alternatives	
Observations	The mechanism with which R2RCbTSP actually determines that there is a discrepancy between expected and actual positions (or, in general, that there is a problem with the shared ride) is a subject of research.
Extensions	

Use case UC15d: Monitor execution of shared ride by TSP (Passenger's side)

Involved high-level functions.

CBTSP, ALM

Description

Table 20 Use case UC15d: Monitor execution of shared ride by TSP (Passenger's side)

Actor	R2Rapp, R2RCbTSP, R2RDrApp
Description	This use case illustrates the interaction that occurs between R2Rapp, R2RCbTSP, and R2RDrApp during the execution of a shared ride taken by R2RPassenger and offered by Driver,

	to generate notifications about potential issues with the shared ride (e.g., delays) that are related to R2RPassenger.
Pre-conditions	R2RPassenger has booked (and paid) an itinerary offer that includes a shared ride offered by a Driver. Tracking of shared ride has been initiated on R2RPassenger's side.
Story	1. R2RCbTSP periodically receives position of R2RPassenger from R2RApp. 2. R2RCbTSP compares current position of R2RPassenger with expected one. 3. If R2RCbTSP notices a discrepancy between R2RPassenger's expected and actual positions, it signals a problem to Driver by sending notification to R2RDrApp. 4. R2RCbTSP updates the Agreement Ledger.
Alternatives	
Observations	As explained in the previous UC, the mechanism with which R2RCbTSP actually determines that there is a discrepancy between expected and actual positions (or, in general, that there is a problem with the shared ride) is a subject of research. In this case, the notification does not go through the S2R ecosystem, because it is not directed to a R2RPassenger, but to an agent (the Driver) of the TSP (hence, it seems out of the scope of the S2R Trip Tracking functions). The actual realization of this UC is not prioritary, and might be left for future work.
Extensions	

Use case UC16: Terminate shared ride and rate experience

This Use case is split in 2 parts: the Passenger's side, and the Driver's side.

Use case UC16a: Terminate shared ride and rate experience (Passenger's side)

Involved high-level functions.

CBTSP, LUP, ALM

Description

Table 21 Use case UC16a: Terminate shared ride and rate experience (Passenger's side)

Actor	R2RPassenger, R2Rapp, R2RCbTSP
Description	This use case illustrates the interaction that occurs between R2RPassenger, R2Rapp, R2RCbTSP when the part of the trip carried out through a shared ride offered by Driver terminates. In particular, R2RPassenger is asked to provide a specific feedback on the Driver (driving style, punctuality, vehicle status, driver behaviour, possible unfair behaviour such as Driver NO SHOW). This Feedback will feed an Ambassador Scoring, namely a rating system in which top ranked App users (both as "Driver" and "Passenger") would receive specific types of awards (e.g., ecobonus, discount on the service payment). This use case details what happens from the side of R2RPassenger. A dual use case (Use case UC16b) details what happens from the point of view of Driver.
Pre-conditions	The shared ride involving Driver and R2RPassenger has ended.
Story	1. R2Rapp detects that the shared ride has ended and notifies R2RPassenger that they can provide feedback on the shared ride. 2. R2RPassenger fills out information about the shared ride: a. Punctuality of Driver (including NO SHOW) b. Diving style c. Conditions of vehicle d. Driver behaviour (e.g., friendliness, quietness) 3. R2Rapp sends information to R2RCbTSP, which updates the information about Driver, in particular the corresponding Ambassador score. 4. R2RCbTSP updates the Agreement Ledger.
Alternatives	
Observations	The feedback might be provided sometime after the shared ride has ended, not necessarily immediately after the completion of the shared ride.
Extensions	• In Step 2a R2Rapp asks if they would like to make the rating publicly available.

	◦ If "yes" in Step 2a, in Step 2b, R2Rapp publishes the experience in the defined channel(s). • In Step 2c "the provided experience/rating" will be sent to the Learning Component to update the preferences model(s).

Use case UC16b: Terminate shared ride and rate experience (Driver's side)

Involved high-level functions.

CBTSP, LUP, ALM

Table 22 Use case UC16b: Terminate shared ride and rate experience (Driver's side)

Actor	R2RDrApp, R2RCbTSP, Driver
Description	This use case illustrates the interaction that occurs between R2RDrApp, R2RCbTSP, and Driver when the part of the trip carried out through a shared ride offered by Driver terminates. In particular, Driver is asked to provide a specific feedback on the R2RPassenger (punctuality, behaviour, flexibility, possible unfair behaviour such as Passenger NO SHOW). This Feedback will feed an Ambassador Scoring, namely a rating system in which top ranked App users (both as "Driver" and "Passenger") would receive specific types of awards (e.g., ecobonus, discount on the service payment). This use case details what happens from the side of Driver. A dual use case (Use case UC16a) details what happens from the point of view of Passenger.
Pre-conditions	The shared ride involving Driver and Passenger has ended.
Story	1. R2RDrApp detects that the shared ride has ended and notifies Driver that they can provide feedback on the shared ride. 2. Driver fills out information about the shared ride: a. Punctuality of R2RPassenger (including NO SHOW) b. R2RPassenger behaviour (e.g., friendliness, quietness)

	3. R2RDrApp sends information to R2RCbTSP, which updates the information about R2RPassenger, in particular the corresponding Ambassador score. 4. R2RCbTSP updates the Agreement Ledger.
Alternatives	
Observations	The feedback might be provided sometime after the shared ride has ended, not necessarily immediately after the completion of the shared ride.
Extensions	• In Step 2a "the provided experience/rating" will be sent to the Learning Component to update the preferences model(s).

Use case UC17: No Show

This Use case is split in 2 parts: the Passenger's side, and the Driver's side.

Use case UC17a: Passenger No Show

Involved high-level functions.

CBTSP, LUP, ALM

Description

Table 23 Use case UC17a: Passenger No Show

Actor	R2RPassenger, R2Rapp, S2R ecosystem, R2RCbTSP, R2RDrApp, Driver
Description	This use case illustrates the interaction that occurs between R2RPassenger, R2Rapp, S2R ecosystem, R2RCbTSP and Driver when R2RPassenger does not join the service at the pick-up point at the agreed time. This use case details what happens from the side of R2RPassenger (i.e., it is the R2RPassenger that does not show up). A dual use case (Use case UC17b) details what happens from the point of view of Driver.
Pre-conditions	Driver has arrived at the pick-up point at the time agreed during booking.
Story	1. R2RPassenger does not join within the agreed amount of time (maximum delay). 2. Driver opens R2RDrApp and inputs information in R2RDrApp that Passenger did not show up.

	3. R2RDrApp notifies R2RCbTSP that R2RPassenger did not show up. 4. R2RCbTSP notifies S2R ecosystem that R2RPassenger did not show up. 5. If necessary, S2R ecosystem notifies R2Rapp (Passenger side) that R2RPassenger did not show up. 6. S2R ecosystem updates the information about R2RPassenger, in particular the corresponding Ambassador score. 7. R2RCbTSP updates the Agreement Ledger.
Alternatives	
Observations	R2RCbTSP might be proactive and, when informed that R2RPassenger did not show up, check the status (e.g., position) of R2RPassenger, assuming that the Trip Tracking function has been activated.
Extensions	

Use case UC17b: Driver No Show

Involved high-level functions.

CBTSP, LUP, ALM

Description

Table 24 Use case UC17b: Driver No Show

Actor	R2RPassenger, R2Rapp, S2R ecosystem, R2RCbTSP, R2RDrApp, Driver
Description	This use case illustrates the interaction that occurs between R2RPassenger, R2Rapp, S2R ecosystem, R2RCbTSP, R2RDrApp and Driver when Driver does not join the service at the pick-up point at the agreed time. This use case details what happens from the side of Driver (i.e., it is the Driver that does not show up). A dual use case (Use case UC17a) details what happens from the point of view of R2RPassenger.
Pre-conditions	R2RPassenger has arrived at the pick-up point at the time agreed during booking.

Story	1. Driver does not join within the agreed amount of time (maximum delay). 2. R2RPassenger opens R2Rapp and inputs information in R2Rapp that Driver did not show up. 3. R2Rapp notifies R2RCbTSP that Driver did not show up. 4. R2RCbTSP notifies S2R ecosystem that Driver did not show up. 5. R2RCbTSP notifies R2RDrApp (Driver side) that Driver did not show up. 6. R2RCbTSP updates the information about Driver, in particular the corresponding Ambassador score, and applies any possible penalty. 7. R2RCbTSP updates the Agreement Ledger.
Alternatives	
Observations	R2RCbTSP might be proactive and, when informed that Driver did not show up, check the status (e.g., position) of Driver, assuming that the Trip Tracking function has been activated. Unlike in Use Case UC17a, the notification to the Driver is sent by R2RCbTSP, because Drivers are outside of the S2R ecosystem.
Extensions	

6.3.7. Passenger-related Learning use cases

This section presents various use cases related to the learning mechanisms that will be implemented by R2Rapp.

To enable R2Rapp to capture the Passengers' needs and attitudes in a context-aware fashion, it should be able to handle three learning mechanisms. Two of the learning mechanisms concern the preferences of Passenger; one concerns the clustering of Passengers having similar behaviours and characteristics.

Use case UC18: Associating Preference Models with newly registered Passenger

Involved high-level functions.

LUP

Description

Table 25 Use case UC18: Associating Preference Models with newly registered Passenger

Actor	R2Rapp
Description	This use case illustrates the high-level steps that occur when R2Rapp triggers its learning module to associate the most promising preference models with a newly registered Passenger. Since R2Rapp does not have any record concerning the new Passenger, to tackle the cold start issue for them, R2Rapp queries the preferences models of the groups(s) that the Passenger belongs to, and associates the preference models of the closest group with the Passenger.
Pre-conditions	R2Rapp has just created a Profile for a new Passenger.
Story	1. R2Rapp triggers its PassengerClustering module and passes the available information from the Passenger as input. 2. PassengerClustering module clusters the Passenger according to the available user groups. 3. PassengerClustering module returns the identifier of the group that the Passenger is closest to (i.e., the best match). 4. Using the identifier returned in Step 3, R2Rapp queries its knowledge models to find the preference model(s) of the group. 5. R2Rapp updates the cloud wallet of the Passenger with the model(s) found in Step 4.
Alternatives	
Observations	
Extensions	

Use case UC19: Learning Passenger’s Contextual Preferences

Involved high-level functions.

LUP

Description

Table 26 Use case UC19: Learning Passenger’s Contextual Preferences

Actor	R2Rapp
Description	This use case illustrates the high-level steps that occur when R2Rapp detects activities by a Passenger and triggers its learning module to update preferences model(s) of the Passenger and the preference model(s) of the groups that the Passenger belongs to.
Pre-conditions	ActivityMonitor module captured some activities by a Passenger.
Story	1. R2Rapp receives the monitored activities from its ActivityMonitor module. 2. R2Rapp fetches the Passenger's record from their cloud wallet. 3. R2Rapp triggers the PassengerPreferenceLearning module and re-learns the model(s) corresponding to the captured activities. 4. R2Rapp updates the Passenger's cloud wallet with the new model(s). 5. R2Rapp fetches the data related to the groups that the Passenger belongs to. 6. R2Rapp triggers the PassengerPreferenceLearning module and re-learn the module(s) corresponding to the captured activities for the groups that the Passenger belongs to. 7. R2Rapp updates its knowledge models with the new model(s) resulted from Step 6.
Alternatives	
Observations	To improve the performance of the component executing the learning algorithms, in Step 5 the learning related to groups can be performed either periodically, or as soon as the number of records reaches a predefined threshold.
Extensions	• In Step 1: Passengers' activities on social media platforms (e.g. Twitter) can be analysed, to better understand their preferences. The monitored activities on social media are the ones which are publicly available and contain specific tags (e.g., mentioning R2R accounts on social media and/or employing pre-defined hashtags).

Use case UC20: Clustering Passengers

Involved high-level functions.

LUP

Description

Table 27 Use case UC20: Clustering Passengers

Actor	R2Rapp
Description	This use case illustrates the high-level steps that occur when R2Rapp detects activities by Passenger and clusters Passengers having the same behaviour/characteristics.
Pre-conditions	The updating time for clustering of Passengers (e.g., the last Saturday of each month at 5 a.m.) is reached. There is at least one new activity record since the last update.
Story	1. R2Rapp triggers its ClusteringPassengers module and passes all the Passenger' activities records (old and new) to it. 2. ClusteringPassenger module, apply the clustering algorithm(s) on the data and clusters Passengers accordingly. 3. R2Rapp will update its knowledge model with the new model of passengers' clusters.
Alternatives	Instead of "at least one new record" as pre-condition, also it is possible to set a threshold for the number of new activities.
Observations	
Extensions	• In Step 2a, ClusteringPassengers module might provide some statistics (as an extra feature) showing the differences between the new clusters and the old one (e.g. the number of groups (clusters) have changed since the last update) if any.

6.3.8. Driver-related Learning use cases

This section presents various use cases related to the learning mechanisms that will be implemented by R2RDrApp.

To enable R2RDrApp to capture the Drivers' needs and attitudes in a context-aware fashion, it should be able to handle three learning mechanisms. Two of the learning mechanisms concern the preferences of Drivers and one concerns the clustering of Drivers having similar behaviours and characteristics.

Use case UC21: Associating Preference Models with newly registered Driver

Involved high-level functions.

LUP

Description

Table 28 Use case UC21: Associating Preference Models with newly registered Driver

Actor	R2RDrApp
Description	This use case illustrates the high-level steps that occur when R2RDrApp triggers its learning module to associate the most promising preference models with a newly registered Driver. Since R2RDrApp does not have any record concerning the new Driver, to tackle the cold start issue for them, R2RDrApp queries the preferences models of the groups(s) that the Driver belongs to, and associates the preference models of the closest group with the Driver.
Pre-conditions	R2RDrApp has just created a Profile for a new Driver.
Story	1. R2RDrApp triggers its DriverClustering module and passes the available information from the Driver as input. 2. DriverClustering module clusters the Driver according to the available Drivers' groups. 3. DriverClustering module returns the identifier of the group that the Driver is closest to (i.e., the best match). 4. Using the identifier returned in Step 3, R2RDrApp queries its knowledge models to find the preference model(s) of the group. 5. R2RDrApp updates the cloud wallet of the Driver with the model(s) found in Step 4.
Alternatives	
Observations	

Extensions	

Use case UC22: Learning Driver's Contextual Preferences

Involved high-level functions.

LUP

Description

Table 29 Use case UC22: Learning Driver's Contextual Preferences

Actor	R2RDrApp
Description	This use case illustrates the high-level steps that occur when R2RDrApp detects activities by a Driver and triggers its learning module to update preferences model(s) of the Driver and the preference model(s) of the groups that the Driver belongs to.
Pre-conditions	ActivityMonitor module captured some activities by a Driver.
Story	1. R2RDrApp receives the monitored activities from its ActivityMonitor module. 2. R2RDrApp fetches the Driver's record from their cloud wallet. 3. R2RDrApp triggers the DriverPreferenceLearning module and re-learns the model(s) corresponding to the captured activities. 4. R2RDrApp updates the Driver's cloud wallet with the new model(s). 5. R2RDrApp fetches the data related to the groups that the Driver belongs to. 6. R2RDrApp triggers the DriverPreferenceLearning module and re-learns the module(s) corresponding to the captured activities for the groups that the Driver belongs to. 7. R2RDrApp updates its knowledge models with the new model(s) resulted from Step 6.
Alternatives	
Observations	To improve the performance of the server responsible for learning purposes. In Step 5, the learning for the groups can

	be done in two ways. Learn periodically or learn as soon as the number of records reaches a predefined threshold.
Extensions	

Use case UC23: Clustering Drivers

Involved high-level functions.

LUP

Description

Table 30 Use case UC23: Clustering Drivers

Actor	R2RDrApp
Description	This use case illustrates the high-level steps that occur when R2RDrApp detects activities by Driver and clusters Drivers having the same behaviour/characteristics.
Pre-conditions	The updating time for clustering of Drivers (e.g., the last Saturday of each month at 5 a.m.) is reached. There is at least one new activity record since the last update.
Story	1. R2RDrApp triggers its ClusteringDriver module and passes all the Driver' activities records (old and new) to it. 2. ClusteringDriver module, apply the clustering algorithm(s) on the data and clusters Drivers accordingly. 3. R2RDrApp will update its knowledge model with the new model of Drivers' clusters.
Alternatives	Instead of "at least one new record" as pre-condition, also it is possible to set a threshold for the number of new activities.
Observations	
Extensions	• In Step 2a, ClusteringDrivers module might provide some statistics (as an extra feature) showing the differences between the new clusters and the old one (e.g. the number of groups (clusters) have changed since the last update) if any.

6.4. Requirements and Interfaces

The following section details the main functions that R2Rapp, R2RDrApp and R2RCbTSP will offer. The following set of requirements is derived using the UCs discussed earlier.

6.4.1. R2RDrApp

Requirement Req1: Driver registration

R2RDrApp will allow GuestUser to register as Driver.

Req1a: During the registration, R2RDrApp will ask GuestUser to provide the mandatory information related to profile such as name, contact information, driving license, etc.

Req1b: R2RDrApp will be able to create Profile for GuestUser as Driver.

Req1c: R2RDrApp will ask Driver to provide optional data regarding the vehicle's facilities (possibly including pictures) such as model, availability of AC, smoking allowed, etc. by means of appropriate notifications.

Req1d: R2RDrApp will ask Driver to provide optional data regarding the personal profile connected preferences such as preferred gender and language of Passenger, etc. by means of appropriate notifications.

Req1e: R2RDrApp will be able to promote terms of the agreement between Driver and Crowd-base TSP for review and agreement by the Driver.

Req1f: R2RDrApp will enable Driver to postpone or ignore providing the optional personal profile connected preferences and vehicle's facilities data.

Req1g: R2RDrApp will allow Driver to specify the importance of the personal profile connected preferences according to appropriate PreferenceImportance fields.

For example, if Driver choose 0 to PreferenceImportance of gender=X, it means that Driver is not willing to accept Passenger whose gender is X.

Req1h: As soon as R2RDrApp receives the Driver's registration, it will be able to trigger DriverClustering module to identify cluster(s) related to the Driver.

Req1i: R2RDrApp will be able to trigger DriverPreferenceLearning module to build a personal preference model for newly registered Driver according to the provided mandatory and optional information.

Req1j: R2Rapp will be able to generate event and send it through appropriate platform (e.g. CIGO) to R2RCbTSP, informing about the new Driver.

Requirement Req2: Incentives settings (Driver's side)

R2RDrApp will enable Driver to enable/disable receiving IncentiveMechanisms by means of choosing from drop-down menu of IncentiveMechanisms.

Requirement Req3: Offering shared ride

R2RDrApp will enable Driver to offer a shared ride.

Req3a: R2RDrApp will allow Driver to provide mandatory parameters (i.e., Travel Data) of the ride.

Req3b: R2RDrApp encourages Driver to provide optional information regarding their vehicle's facilities by means of appropriate notification while offering a shared ride if already not provided.

Req3c: R2RDrApp encourages Driver to provide optional information regarding their personal preferences by means of appropriate notification while offering a shared ride if already not provided.

Req3d: R2RDrApp will ask some external services (e.g., Google Maps) about the path and driver might modify the path returned by the external service.

Req3e: R2RDrApp will enable Driver to specify the number and/or location(s) of possible stop(s).

Req3f: R2RDrApp will enable Driver to make the shared ride periodically, repeated within a user-defined period or specific dates.

Requirement Req4: Offer-related communications between R2RDrApp and R2RCbTSP

R2RDrApp will be able to generate events regarding offering, modification and cancellation of rides and send it through appropriate platform (e.g. CIGO) to R2RCbTSP.

Requirement Req5: Rides' duration and distance computations

R2RDrApp will be able to ask an external service (e.g. Google Maps) for computing the distance and time of offered rides.

Requirement Req6: Cancelling shared rides

R2RDrApp will enable Driver to cancel their offered ride(s) and optionally indicate the reason.

Contract No. 881825

Req6a: Upon cancelation of the offered ride(s), R2RDrApp will be able to update Driver's Reliability score.

Req6b: Upon the cancelation of an offered ride, R2RDrApp will be able to generate an event and send it through an appropriate platform (e.g. CIGO) to S2R ecosystem to inform the Passenger(s) who booked the ride.

Requirement Req7: Modifying shared rides

R2RDrApp will enable Drivers to modify their offered ride(s).

Req7a: R2RDrApp will be able to send a request to R2RCbTSP through an appropriate platform (e.g., CIGO) and check if the offered ride(s) is booked by any Passenger or not.

Req7b: In case of requesting for modification of the booked shared rides, before consolidating the modifications, R2RDrApp will enable the Driver to optionally ask the Passenger(s)' permission(s) for the modification.

Req7c: Upon the permission request by the Driver, R2RDrApp will be able to generate an event and notify R2RDrTSP through appropriate platform (e.g., CIGO!) to send the permission request to the Passenger(s).

Req7d: Upon the modification of offered ride(s), R2RDrApp will be able to generate an event and notify R2RDrTSP through appropriate platform (e.g., CIGO!) to notify Passenger(s) who booked those offers.

Requirement Req8: Profile update (Driver's side)

R2RDrApp will enable Drivers to update their optional personal information (job, salary, …) and profile connected preferences (e.g. preferred language of the passengers, etc.) and their vehicle's facilities via appropriate menus.

Requirement Req9: Activities Monitor (Driver's side)

R2RDrApp will include an ActivityMonitor module to monitor Driver's activities for learning purposes. This module will fulfil the following requirements.

Req9a: Monitoring the creation of a Drivers' profile or the modification on an existing one.

Req9b: Monitoring the content with which Drivers are interacting, to determine the context of the content. More precisely, the content can be anything that is shown to Drivers – for example, an article about the new services in the region, information regarding possible passengers for a shared ride and so on. This requirement is subject to availability of such monitoring functions in the existing TC and respecting the GDPR.

Contract No. 881825

Req9c: Monitoring which IncentiveMechanisms are (De)activated by Drivers.

Req9d: Monitoring which Passenger(s) are approved by each Driver.

Req9e: Monitoring the activities regarding offering/modification/cancelling rides.

Req9f: Monitoring the feedbacks provided by Drivers and for the Drivers.

Requirement Req10: Preferences learning (Driver's side)

R2RDrApp will implement DriverPreferenceLearning function which will be able to receive the monitored activities of the Drivers and learn Drivers' preferences and update their knowledge model(s) accordingly.

Requirement Req11: Location personalization (Driver's side)

R2RDrApp will enable Drivers to provide names for specific locations (Home, Office, ...) to facilitate their requests and enhance learning the contextual preferences.

Requirement Req12: Feedback about Passenger

R2RDrApp will provide a Service Feedback functionality for Drivers to rate the Passengers through predefined questions.

Requirement Req13: Service Feedback management (Driver's Side)

R2RDrApp will enable Drivers to make their received/provided feedbacks publicly available or private to the scope of S2R ecosystem for learning purposes through appropriate notification for both side of the feedback (provider and receiver).

Requirement Req14: Tracking the Drivers involved in a shared ride

R2RDrApp will be able to detect the position of the Driver for logistic purposes.

Req14a: R2RDrApp will be able to generate events about the location of the Driver and send it to R2RCbTSP through appropriate platform (e.g., CIGO!).

Requirement Req15: Shared ride tokens management (Driver's side)

R2RDrApp will be able to store the shared rides tokens on the Driver's personal application to be used for validation.

Requirement Req16: Shared ride execution management

R2RDrApp will enable Drivers to declare the start and ending of their shared rides.

Req16a: Upon the execution of the rides by Drivers, R2RDrApp will be able to generate event and through appropriate platform (e.g. CIGO) send it to R2RCbTSP.

Requirement Req17: Compensation of Driver management (Driver's side)

Upon receiving events regarding the compensation(s) of the Driver, R2RDrApp will be able to update their profile with the confirmed compensation(s) and notify them by means of appropriate notification.

6.4.2. R2Rapp

Requirement Req18: Passenger registration

R2Rapp will allow GuestUser to register as Passenger.

Req18a: During the registration, R2Rapp will ask GuestUser to provide the mandatory information related to profile such as name, contact information, etc.

Req18b: R2Rapp will be able to create Profile for GuestUser as Passenger.

Req18c: R2Rapp will ask Passenger to provide optional data regarding the personal profile connected preferences such as preferred means of transportation, possible health issues, etc. by means of appropriate notifications.

Req18d: R2Rapp will enable Passenger to postpone or ignore providing the optional personal profile connected preferences.

Req18e: R2Rapp will allow Passenger to specify the importance of the personal profile connected preferences according to appropriate PreferenceImportance fields.

For example, if Passenger chooses 0 to PreferenceImportance of transportationMode=Bike, it means that Passenger is not willing to accept offers that include Bike as a transportation mode.

Req18f: As soon as R2Rapp receives the Passenger's registration, it will be able to trigger PassengerClustering module to identify related cluster(s) to the passenger.

Req18g: R2Rapp will be able to trigger PassengerPreferenceLearning module to build a personal preference model for newly registered Passenger according to the provided mandatory and optional information.

Req18h: R2Rapp will be able to promote terms of the agreement between Passenger and Crowd-base TSP for review and agreement by the Passenger.

Requirement Req19: Incentives settings (Passenger's side)

R2Rapp will enable Passenger to enable/disable receiving IncentiveMechanisms by means of choosing from drop-down menu of IncentiveMechanisms.

Requirement Req20: Incentivizing Passengers to become Drivers

R2Rapp will be to incentivize Passengers to become Drivers through appropriate notification.

Req20a: R2Rapp will enable Passengers to register, postpone or ignore becoming Drivers.

Req20b: Upon acceptance of becoming a Driver by a Passenger, R2Rapp will be able to generate an event and pass the Passengers' profile data to the S2R ecosystem through an appropriate platform (e.g. CIGO) to be delivered to R2RDrApp for autocomplete the data whenever available from the Passenger's profile and the rest of the procedure will be handled by R2RDrApp (see Requirement Req1).

Req20c: R2Rapp will enable Passengers to share their vehicles if part of their selected itinerary includes a ride.

Requirement Req21: Promoting the search options during mobility request

During a mobility request, R2Rapp will promote the optional search options by means of appropriate notification in order to capture better the contextual preferences of the Passenger.

Req21a: During a mobility request, R2Rapp will allow the Passenger to specify optional contextual preferences for an available list of services (Search options) by indicating the PreferenceImportance score from 0 to 5. 0 shows excluding the offers containing those services and 5 means the highest score.

Requirement Req22: Profile update (Passenger's side)

R2Rapp will enable Passengers to update their optional personal information (job, salary, …) and profile connected preferences (e.g. preferred transportation mode, comfort class, etc.) via appropriate menus.

Requirement Req23: Ranking and Categorizing offers

Given a list of offers, R2Rapp will be able to present travel solutions in a ranked manner highlighting for each solution the incentives.

Req23a: R2Rapp will be able to assign appropriate category label(s) – i.e. offer categories (such as Quick, Short, Reliable, etc.) – to each of the received offers through its OfferCategorizer module.

Req23b: R2Rapp will be able to provide score between 0 and 1 for each offer according to the Passenger contextual preferences through its OfferRanker module.

Contract No. 881825

Req23c: R2Rapp will be able to invoke IncentiveProviders through an appropriate platform and ask them to apply IncentiveMechanisms on the offers.

Req23d: R2Rapp will enable Passenger to provide further preferences and order the presented ranked offers by means of an appropriate menu.

Requirement Req24: Activities Monitor (Passenger's side)

R2Rapp will include an ActivityMonitor module to monitor Passenger's activities for learning purposes specifically:

Req24a: Monitoring the creation of a Passenger's profile or modification on an existing one.

Req24b: Monitoring the content with which Passengers are interacting, to determine the context of the content. The content can be anything that is shown to Passengers, such as an article about the new services in the region, information regarding the offers and so on. This requirement is subject to availability of such monitoring functions in the existing TC and respecting the GDPR.

Req24c: Monitoring the set of ranked offers presented to the Passenger and the booked one.

Req24d: Monitoring which IncentiveMechanisms are (de)activated by Passengers. For example, a Passenger might deactivate applying IncentiveMechanisms concerning eco-friendly options.

Requirement Req25: Preferences learning (Passenger's side)

R2Rapp will implement PassengerPreferenceLearning function which will be able to receive the monitored activities of the Passenger and learn Passengers' preferences and update their knowledge model(s) accordingly.

Requirement Req26: Incentive enricher

R2Rapp will be able to generate events to invoke IncentiveProviders through appropriate platforms (e.g. CIGO) to enrich the offers by IncentiveMechanisms.

Requirement Req27: Location personalization (Passenger's side)

R2Rapp will enable Passengers to provide names for specific locations (Home, Office, …) to facilitate their requests and enhance learning the contextual preferences.

Contract No. 881825

Requirement Req28: Service Feedback

R2Rapp will provide a Service Feedback functionality for Passengers to rate various services (both perceptible and non-perceptible by touch) they received during a trip.

Req28a: R2Rapp will provide a Service Feedback functionality for Passengers to rate Drivers by means of predefined questions.

Req28b: R2Rapp will provide a Service Feedback functionality for Passengers to rate each received service they received by means of predefined questions.

Req28c: R2Rapp will enable the Passenger(s) to declare if the Driver did not show up according to the schedule ("Diver No Show").

Requirement Req29: Service Feedback management (Passenger's Side)

R2Rapp will enable Passengers to make their received/provided feedbacks publicly available or private to the scope of S2R ecosystem for learning purposes for both side of the feedback (provider and receiver).

Requirement Req30: Tracking the Passengers involved in a shared ride

R2Rapp will be able to detect the position of the Passenger(s) for logistic purposes.

Req30a: R2Rapp will be able to generate events regarding the location of the Passenger(s) and send it to R2RCbTSP through appropriate platform (e.g., CIGO!).

Req30b: R2RDrApp will enable Passengers to activate sharing their positions with specified TC users for socializing.

Requirement Req31: Shared ride tokens management (Passenger's side)

R2Rapp will be able to store the shared rides tokens on the Passenger's personal application to be used for validation.

6.4.3. R2RCbTSP

Requirement Req32: Compensation of Driver management (TSP's side)

R2RCbTSP will be able to manage the events and computations concerning the compensation of Drivers.

Contract No. 881825

Req32a: R2RCbTSP will be able to receive events from the S2R ecosystem regarding the selection of shared rides by passengers through appropriate platform (e.g., CIGO!).

Req32b: R2RCbTSP will be able to compute the compensations for Drivers whose rides are booked.

Req32c: R2RCbTSP will update the Driver profile with the computed compensations.

Req32d: R2RCbTSP will be able to generate events to notify S2R ecosystem and R2Rapp through appropriate platform (e.g., CIGO!) concerning the confirmation of the applied incentives for the Drivers.

Req32e: R2RCbTSP will be able to update the Agreement Ledger about the provided compensation(s).

Requirement Req33: Shared ride retrieval

R2RCbTSP will provide an API to communicate with S2R ecosystem regarding shared rides.

Req33a: R2RCbTSP will provide a function to inform S2R ecosystem about the regions covered by R2RCbTSP.

Req33b: R2RCbTSP will be able to provide rides from the covered regions for specific Travel Episodes upon request (e.g., for a given date and area (source, destination)).

Requirement Req34: Shared ride execution monitoring

Req34a: R2RCbTSP will be able to detect when the rides start and end.

Req34b: R2RCbTSP will be able to detect the positions of Driver and Passenger(s) involved in a shared ride for monitoring purposes.

Req34c: Given the current location of the Driver/Passenger(s) involved in a shared ride, R2RCbTSP will be able to ask external services (e.g. Google Maps) to compute the time which they require to reach the starting location.

Req34d: R2RCbTSP will be able to publish events regarding the execution of the ride (e.g. Notifying the Passenger(s) that Driver will be late for 10 minutes).

Req34e: R2RCbTSP will be able to update the Agreement Ledger about the execution of the rides.

Requirement Req35: Shared ride tokens generation

R2RCbTSP will be able to generate tokens (e.g., QR code) for the confirmed shared rides.

Req35a: R2RCbTSP will be able to update the Agreement Ledger upon generation of token(s).

Requirement Req36: Drivers and Passengers management

R2RCbTSP will be able to manage the information of the Passenger(s) and Driver involved in a shared ride.

Req36a: R2RCbTSP will be able to provide an API to add/update the information about Drivers.

Req36b: R2RCbTSP will be able to add/remove Passenger(s) to a shared ride.

Req36c: Upon any changes in the information of the Passenger(s) and Driver involved in a shared ride (Insertion, Modification, etc.) R2RCbTSP will be able to update the Agreement Ledger.

Requirement Req37: Agreement Ledger management

R2RCbTSP should be able to update the trip information in case of events related to this trip.

Req37a: Upon the cancelation of a shared-ride which is booked by one or more Passengers, the R2RCbTSP should be able to update the Agreement Ledger with the cancelled ride.

Req37b: Upon the modification of a shared-ride which is booked by one or more Passengers, the R2RCbTSP should be able to update the Agreement Ledger with the modified ride.

Req37c: Upon the selection of a shared-ride by one a Passenger, the R2RCbTSP should be able to update the Agreement Ledger with the selected ride.

Req37d: Upon the computation of the compensation of the shared-ride, the R2RCbTSP should be able to update the Agreement Ledger with the compensation of the Driver.

Req37e: Upon the generation of the ride's token, the R2RCbTSP should be able to update the Agreement Ledger with the generated token.

Contract No. 881825

Req37f: Upon the validation of the shared ride (S2R has already flagged the ride "Started"), the R2RCbTSP should be able to update the Agreement Ledger with the validated ride.

Req37g: Upon the receipt of the significant events (e.g. Delay) from the trip tracker, the R2RCbTSP should be able to update the Agreement Ledger with the event.

Req37h: Upon the detection of the discrepancy between Driver's expected and actual positions (by R2RCbTSP), the R2RCbTSP should be able to update the Agreement Ledger with the issue.

Req37i: Upon the detection of the discrepancy between Passenger's expected and actual positions (by R2RCbTSP), the R2RCbTSP should be able to update the Agreement Ledger with the issue.

Req37j: Upon the termination of the ride receipt by the Passenger, the R2RCbTSP should be able to update the Agreement Ledger with the termination state.

Req37k: Upon the termination of the ride receipt by the Driver, the R2RCbTSP should be able to update the Agreement Ledger with the termination state.

Req37l: Upon the receipt of the Passenger No Show, the R2RCbTSP should be able to update the Agreement Ledger with the Passenger No Show.

Req37m: Upon the receipt of the Driver No Show, the R2RCbTSP should be able to update the Agreement Ledger with the Driver No Show.

Contract No. 881825

6.4.4. Requirements coverage and priority analysis

The matrix presented below details how individual requirements introduced earlier cover the use cases. Moreover, the left column of the table specifies each requirement's priority (PRI) in the scale of High, Medium, and Low priority denoted by H, M, and L, respectively.

Table 31: The coverage of use cases by requirements

PRI		UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19	UC20	UC21	UC22	UC23
H	Req1	X	X							X	X													
L	Req2	X	X							X	X													
H	Req3			X	X					X	X													
H	Req4			X	X					X	X	X												
M	Req5			X	X					X	X													
H	Req6					X												X						
M	Req7						X																	
H	Req8	X	X							X	X													
M	Req9	X	X	X	X	X	X			X	X						X	X				X	X	X
H	Req10	X	X	X	X	X	X			X	X						X	X				X	X	X
L	Req11			X	X					X	X													
M	Req12																X	X						
M	Req13																X	X						
H	Req14																							
H	Req15													X	X	X								
H	Req16														X		X	X						
M	Req17																							
L	Req18																							
L	Req19		X					X	X	X	X													
H	Req20		X							X	X													
L	Req21							X	X			X												
L	Req22																							
H	Req23							X	X	X	X	X												
M	Req24		X					X	X	X	X	X					X	X	X	X	X			
H	Req25							X	X	X	X						X	X	X	X	X			
H	Req26							X	X	X	X													
L	Req27							X	X	X	X													
M	Req28																X	X						
M	Req29																X	X						
M	Req30															X	X	X						
H	Req31													X	X									
H	Req32					X	X						X											
H	Req33			X	X			X		X	X	X												
H	Req34											X			X	X	X	X						
H	Req35													X	X									
H	Req36	X	X			X	X			X	X		X	X	X	X	X	X						
M	Req37					X	X					X	X	X	X	X	X	X						

7. SPECIFICATION

This section provides a specification of the main elements of the R2R system, given in terms of UML diagrams.

The elements are separated in two groups. Those related to the interaction with the users (i.e., R2Rapp and R2RDrApp), and those related to the Crowd-based TSP (i.e., R2RCbTSP). An overview of these components is provided first, then a few Sequence Diagrams that describe some significant interactions (including some occurring with the S2R ecosystem) is shown.

In this deliverable, only the main features of the UML model are highlighted. The complete UML model can be found in the annex. The complete UML model is to be considered a "living artifact", in the sense that, as the components and mechanisms identified in the model are developed, their design might undergo modifications and adjustments, which should be reflected in their UML representation.

7.1. Main concepts

The modules and services that will be developed in the R2R project rely on various concepts such as rides, preferences, facilities of vehicles, etc. For reasons of brevity, this deliverable does not present all such concepts, which can be found in the complete UML model.[2] Figure 4 and Figure 5, instead, present a few of the most relevant ones. In particular, Figure 4 shows classes capturing the features of rides and of enriched itinerary offers.

[2] The UML model has been originally created for Deliverable D2.3 [1] taking account the terms defined in the MaaSive glossary, the analysis carried out in Deliverable D2.1 [3] on terminology alignment, and the study carried out in Deliverable D2.2 [2]. The revision of the model carried out in this deliverable takes into account the finalizations of the concepts presented in Delieverable D2.4 [5] and Deliverable D2.5 [6].

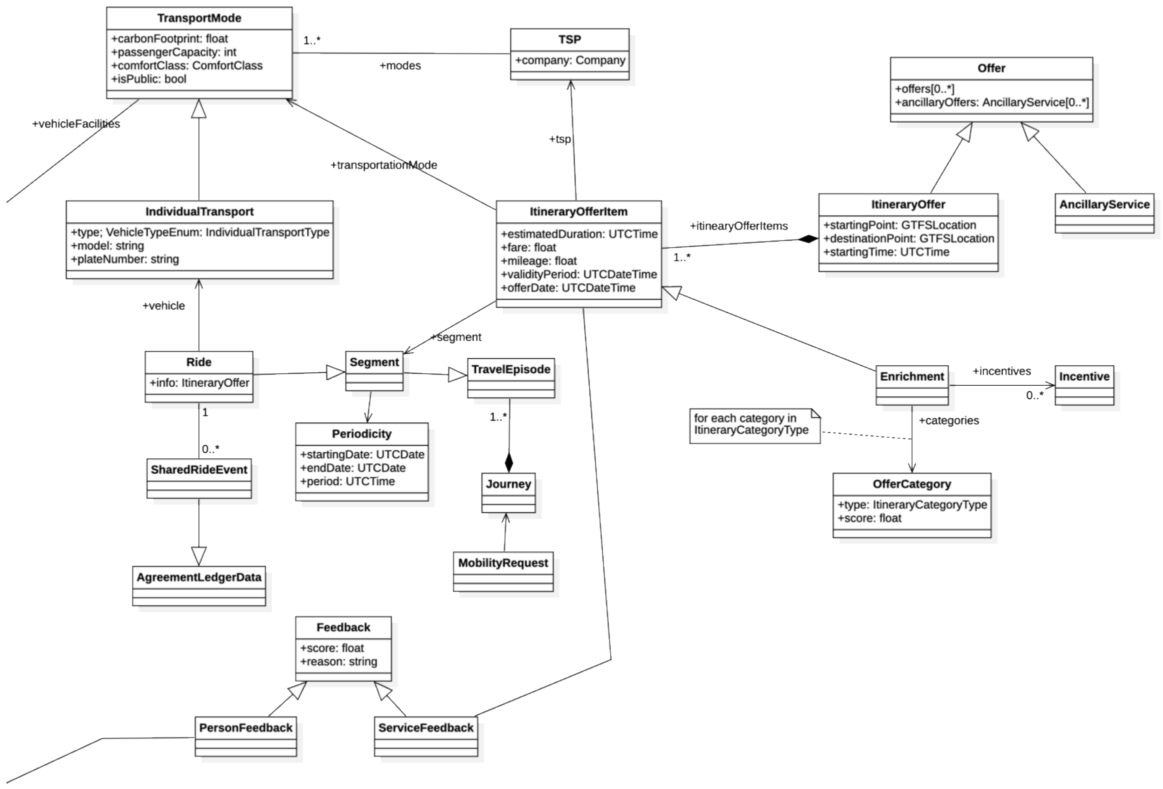

Figure 4 – Snippet of UML Class Diagram representing key concepts for the R2R system (rides).

As the figure shows, a ride is a kind of segment, so it can be part of an itinerary offer (where the notions of segment and of itinerary offer are defined in the MaaSive glossary). An enrichment of an itinerary offer, instead, extends the notion of itinerary offer with features such as the category of its itinerary offer items, and the incentives (if any) associated with them.

Figure 5 shows the type of facilities that a private vehicle can have, and the types of items that one might be allowed to carry on a shared ride.

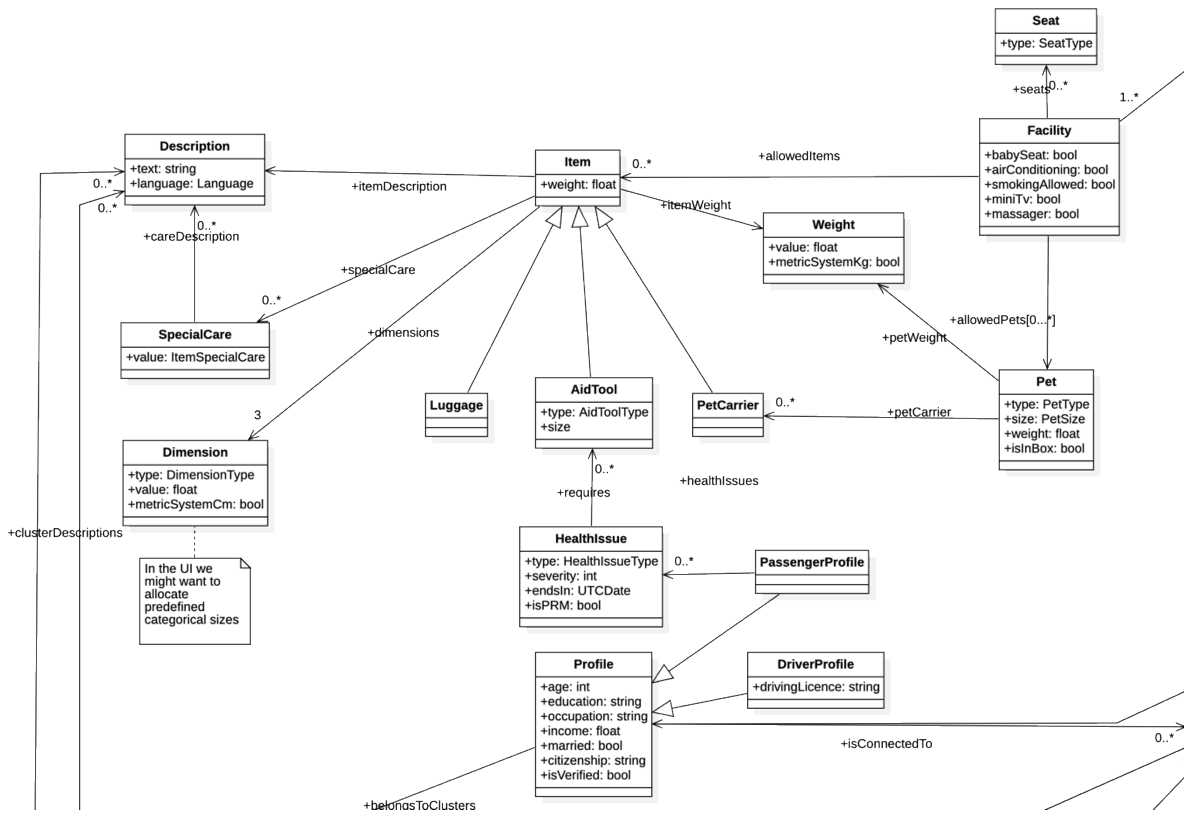

Figure 5 – Snippet of UML Class Diagram representing key concepts for the R2R system (items).

7.2. Components

Figure 6 and Figure 7 show the main elements to be developed in the R2R project that will, on one hand, enhance the experience of Passengers using the S2R ecosystem and, on the other hand, allow Drivers to interact with the Crowd-based TSP. In particular, the figure shows that users, depending on whether their role is that of Driver or of Passenger, interact with two different modules, the R2R Travel Companion (i.e., the R2Rapp identified in Section 6.2), or the R2R Driver Companion (i.e., the R2RDrApp of Section 6.2). These modules allow users to interact with the system, for example to create and modify rides (on the Driver's side), or to get offers. Notice that the functions offered to Passengers (e.g., retrieving offers) are already part of the TC developed in previous S2R projects, but they are also presented here because their behaviour will be modified by the extensions developed within the R2R project.

R2Rapp and R2RDrApp could be developed as part of the same application, or as two separate applications. See Section 7.4 and Section 8.5 for a discussion on this topic and on the feasibility of the approaches.

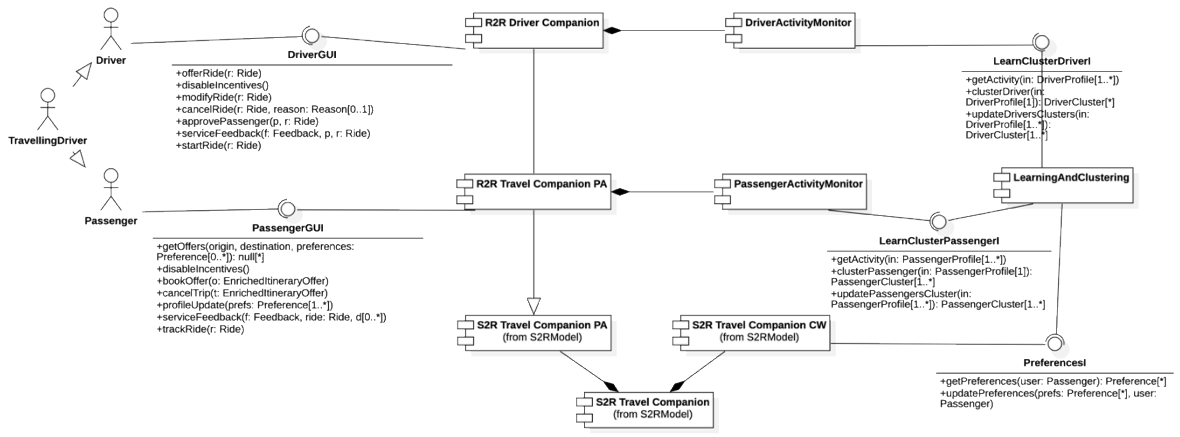
Driver
TravellingDriver
Passenger
DriverGUI
+offerRide(r: Ride)
+disableIncentives()
+modifyRide(r: Ride)
+cancelRide(r: Ride, reason: Reason[0..1])
+approvePassenger(p, r: Ride)
+serviceFeedback(f: Feedback, p, r: Ride)
+startRide(r: Ride)
PassengerGUI
+getOffers(origin, destination, preferences: Preference[0..*]): null[*]
+disableIncentives()
+bookOffer(o: EnrichedItineraryOffer)
+cancelTrip(t: EnrichedItineraryOffer)
+profileUpdate(prefs: Preference[1..*])
+serviceFeedback(f: Feedback, ride: Ride, d[0..*])
+trackRide(r: Ride)
R2R Driver Companion
R2R Travel Companion PA
S2R Travel Companion PA (from S2RModel)
S2R Travel Companion CW (from S2RModel)
S2R Travel Companion (from S2RModel)
DriverActivityMonitor
PassengerActivityMonitor
LearningAndClustering
LearnClusterDriverI
+getActivity(in: DriverProfile[1..*])
+clusterDriver(in: DriverProfile[1]): DriverCluster[*]
+updateDriversClusters(in: DriverProfile[1..*]): DriverCluster[1..*]
LearnClusterPassengerI
+getActivity(in: PassengerProfile[1..*])
+clusterPassenger(in: PassengerProfile[1]): PassengerCluster[1..*]
+updatePassengersCluster(in: PassengerProfile[1..*]): PassengerCluster[1..*]
PreferencesI
+getPreferences(user: Passenger): Preference[*]
+updatePreferences(prefs: Preference[*], user: Passenger)

Figure 6 – Main R2R components interacting with users (Drivers and Passengers).

As Figure 6 shows, the R2R project will develop modules (*DriverActivityMonitor* and *PassengerActivityMonitor*) for monitoring the activity of users who are interacting with the system through the R2Rapp and R2RDrApp, which will feed another module (the *LearningandClustering* module) which will use this information to update the user preferences.

It is envisaged (see also Section 7.4) that the learning component will not be created as parts of the TC, but it will be provided through services reachable through suitable endpoints. The activity monitors, instead, must necessarily be part of the TC.

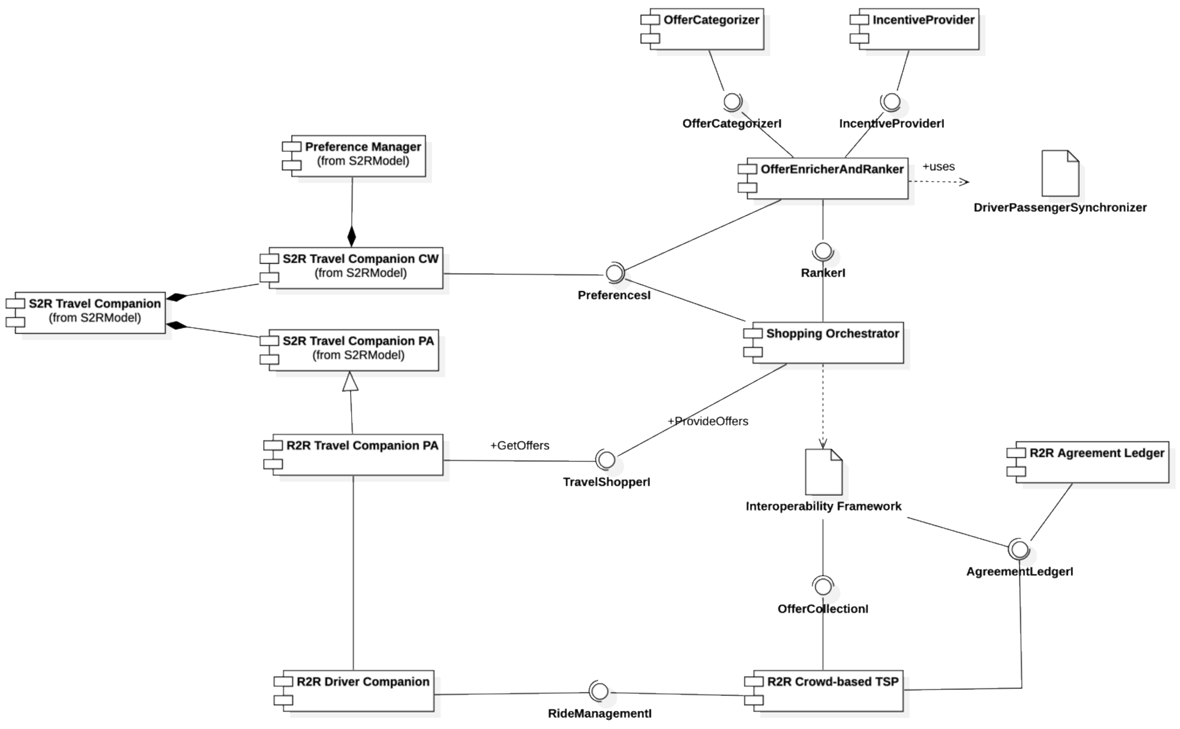

Figure 7 – Main R2R components interacting with S2R ecosystem.

In addition, a module for the enrichment and ranking of itinerary offers will be provided (see Figure 7). This module will exploit other components to perform various functions:

- categorize offers (through the *OfferCategorizer* module)
- add incentives to offers (through the *IncentiveProvider* module, which corresponds to the "INCENTIVE PROVIDER" module mentioned in Deliverable D2.1)
- find the best matches between Drivers and Passengers (*DriverPassengerSynchronizer* element).

As described in UC7a, the offer ranker and associated modules will be invoked by the S2R ecosystem (and in particular by the Shopping Orchestrator), so they will be provided through services reachable through suitable endpoints.

Finally, the R2R project will develop the *Agreement Ledger* module, which will allow for the safe and secure storage of relevant information of segments, including shared rides (notice that it is envisaged that the module can be used not only by the Crowd-based TSP, but also by other elements of the S2R ecosystem).

Contract No. 881825

Figure 8 shows the interfaces of the module realizing the Crowd-based TSP (i.e., R2RCbTSP). The functions are those typical of a TSP and concern the creation and management of shared rides: addition and cancellation of rides, retrieval of rides covering a certain path, ride booking, ride validation, ride monitoring. In addition, the R2RCbTSP supports the management of Drivers, and, in particular, their registration and the management of their preferences. The figure also shows that R2RCbTSP interacts with the R2R Agreement Ledger to keep track of relevant information concerning the ride.

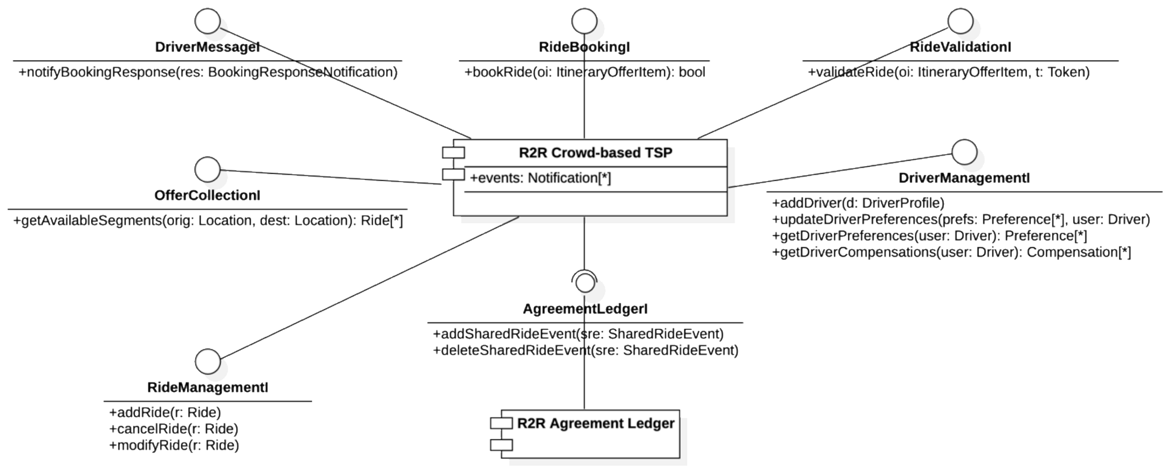
DriverMessageI
+notifyBookingResponse(res: BookingResponseNotification)
RideBookingI
+bookRide(oi: ItineraryOfferItem): bool
RideValidationI
+validateRide(oi: ItineraryOfferItem, t: Token)
R2R Crowd-based TSP
+events: Notification[*]
OfferCollectionI
+getAvailableSegments(orig: Location, dest: Location): Ride[*]
DriverManagementI
+addDriver(d: DriverProfile)
+updateDriverPreferences(prefs: Preference[*], user: Driver)
+getDriverPreferences(user: Driver): Preference[*]
+getDriverCompensations(user: Driver): Compensation[*]
AgreementLedgerI
+addSharedRideEvent(sre: SharedRideEvent)
+deleteSharedRideEvent(sre: SharedRideEvent)
RideManagementI
+addRide(r: Ride)
+cancelRide(r: Ride)
+modifyRide(r: Ride)
R2R Agreement Ledger

Figure 8 – Interfaces of the Crowd-based TSP.

To handle the monitoring of rides, the R2R system will provide a module, R2R Ride Tracker, which is depicted in Figure 9. The module should closely interact with (and possibly be part of) the R2RCbTSP. In particular, it should provide an interface, *RideMonitoringI* that enables interested parties to subscribe to notifications concerning relevant impacts[3] (e.g., delays, cancellations) on journeys caused by events regarding the chosen ride. Indeed, the R2R Ride Tracker performs the functions of a partial Trip Tracker (see the MaaSive glossary for the definition of partial Trip Tracker), so it receives events from event sources (for example the position of the Driver sent by R2RDrApp) and processes them. If it detects that there is some impact on the shared ride, then it notifies any subscriber to that ride

[3] For the notion of "impact" see the MaaSive glossary.

(see also Figure 13 and associated description) and it uses the ALM to store the alarm event in the ledger.

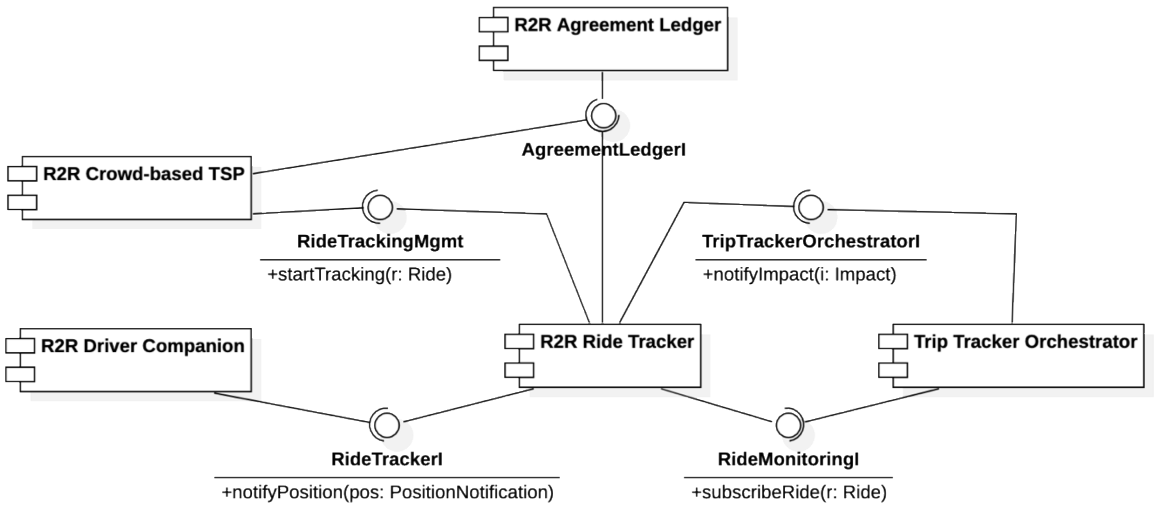

Figure 9 – Components involved in ride tracking.

7.3. Interactions

This section presents some UML Sequence Diagrams that describe the most relevant interactions among the components identified in Section 7.2.

Figure 10 depicts the interaction that occurs between Driver, R2RDrApp and R2RCbTSP when a new shared ride is created. It essentially corresponds to use cases UC3 and UC4.

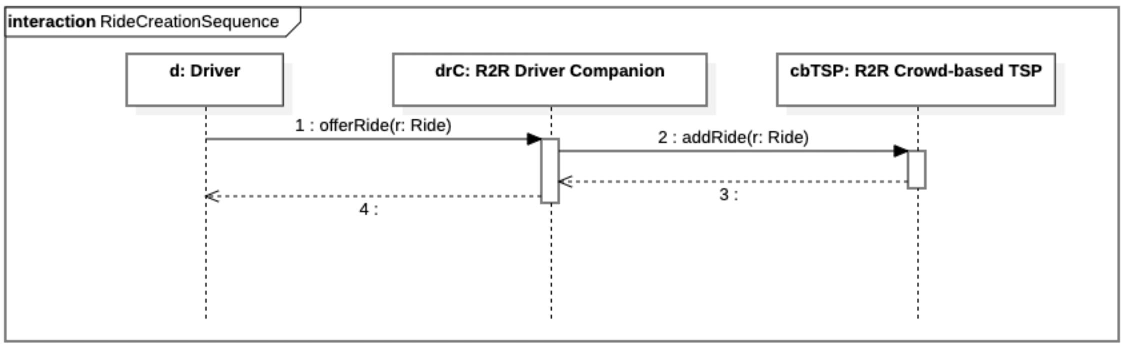
interaction RideCreationSequence
d: Driver
drC: R2R Driver Companion
cbTSP: R2R Crowd-based TSP
1 : offerRide(r: Ride)
2 : addRide(r: Ride)
3 :
4 :

Figure 10 – Sequence Diagram capturing the interaction concerning the creation of a shared ride.

Figure 11, instead, describes the interaction that occurs when a Passenger asks the R2Rapp to retrieve offers for a trip. It essentially corresponds to use cases UC7, UC7a, and UC7b. Notice that the interaction also involves some components from the S2R ecosystem, and in particular the *Shopping Orchestrator*[4] and the *Interoperability Framework*. These interactions are not fully detailed here, because they occur outside of the scope of the R2R project, but they have been included nonetheless to provide a fuller picture of the interaction.

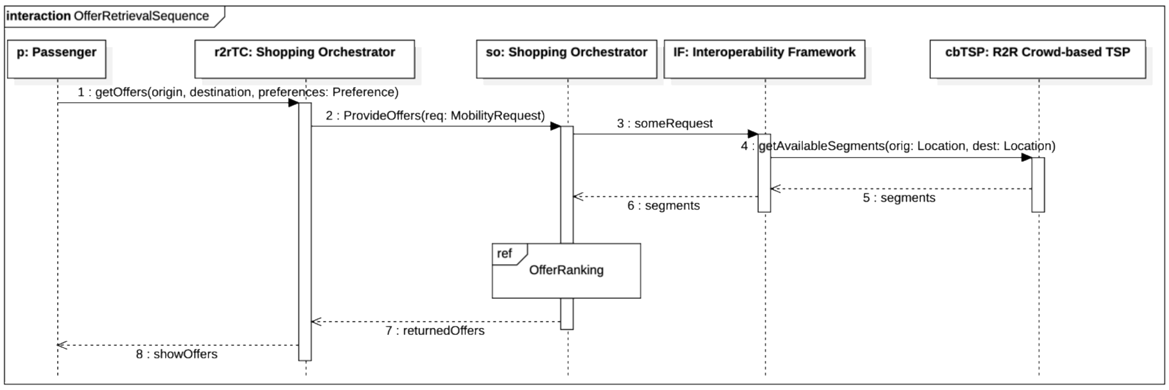
interaction OfferRetrievalSequence
p: Passenger
r2rTC: Shopping Orchestrator
so: Shopping Orchestrator
IF: Interoperability Framework
cbTSP: R2R Crowd-based TSP
1 : getOffers(origin, destination, preferences: Preference)
2 : ProvideOffers(req: MobilityRequest)
3 : someRequest
4 : getAvailableSegments(orig: Location, dest: Location)
5 : segments
6 : segments
ref
OfferRanking
7 : returnedOffers
8 : showOffers

Figure 11 – Sequence Diagram capturing the interaction concerning the retrieval of itinerary offers (including their enrichment and ranking).

Part of the process for offer building by the S2R ecosystem concerns the ranking of offers (mentioned in use case UC7a). Figure 12 details the interaction among the components (*OfferEnricherAndRanker*, *OfferCategorizer*, *IncentiveProvider*) that perform the enrichment and the ranking of the offers.

[4] Notice that, as remarked in Deliverable D2.1, the notion of "Shopping Orchestrator" in the latest version of the MaaSive glossary has been replaced with the notion of "Travel Solution Aggregator". However, the software architecture of the S2R ecosystem (see also Figure 15 in Section 8.1) still refers to "orchestrators", so the latter term is used in the UML models, which focuses on the software components of the R2R system.

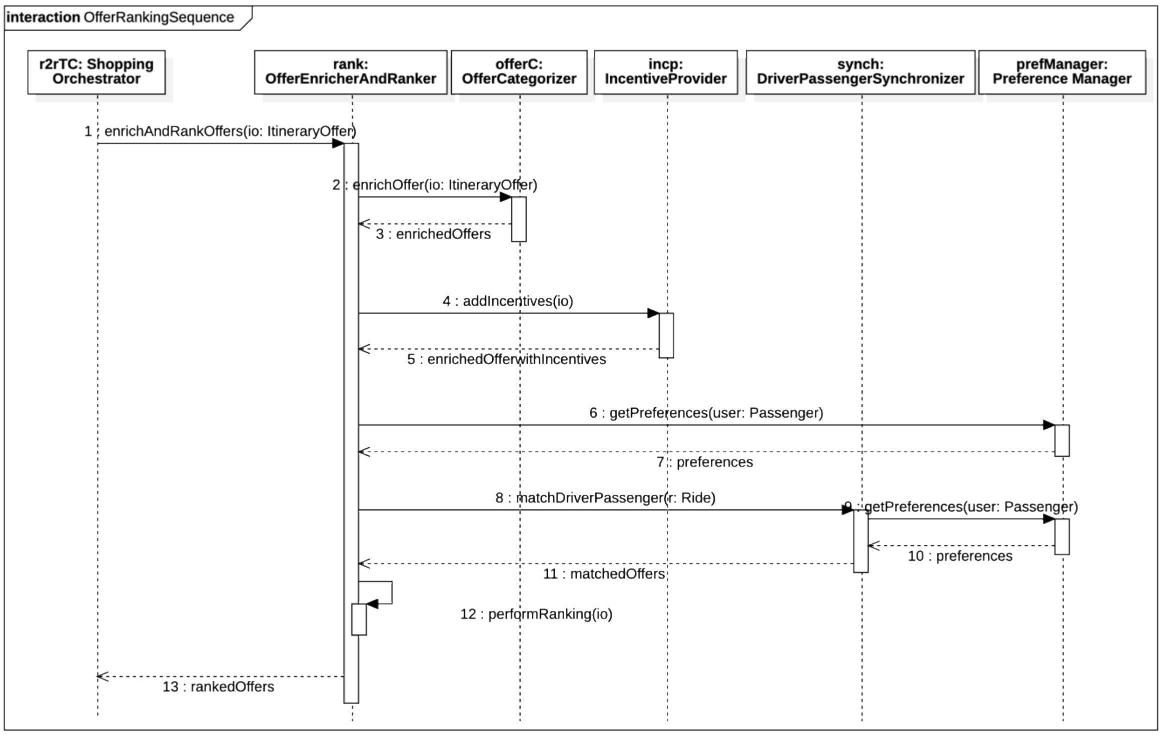
interaction OfferRankingSequence
r2rTC: Shopping Orchestrator
rank: OfferEnricherAndRanker
offerC: OfferCategorizer
incp: IncentiveProvider
synch: DriverPassengerSynchronizer
prefManager: Preference Manager
1 : enrichAndRankOffers(io: ItineraryOffer)
2 : enrichOffer(io: ItineraryOffer)
3 : enrichedOffers
4 : addIncentives(io)
5 : enrichedOfferwithIncentives
6 : getPreferences(user: Passenger)
7 : preferences
8 : matchDriverPassenger(r: Ride)
9 : getPreferences(user: Passenger)
10 : preferences
11 : matchedOffers
12 : performRanking(io)
13 : rankedOffers

Figure 12 – Sequence Diagram capturing the detail of the interaction concerning the enrichment and ranking of itinerary offers.

Figure 13 depicts the interaction that occurs between R2RDrApp, R2RCbTSP and the S2R ecosystem (in particular, the Trip Tracker Orchestrator) when a shared ride is monitored (this essentially corresponds to use cases UC15a, minus the termination of the ride, and UC15c). As the diagram shows, when the R2RCbTSP receives the position of the Driver, it determines whether the shared ride is running smoothly or not; in the latter case (in which an "impact" is determined on the shared ride), the S2R ecosystem, and in particular the Trip Tracker Orchestrator, is notified of the impact. According to the architecture of the S2R ecosystem, for the Trip Tracking Orchestrator to be able to receive the notification of the impact, it must have previously subscribed to the feed of impacts relevant to the shared ride (this step is not shown in the Sequence Diagram).

Contract No. 881825

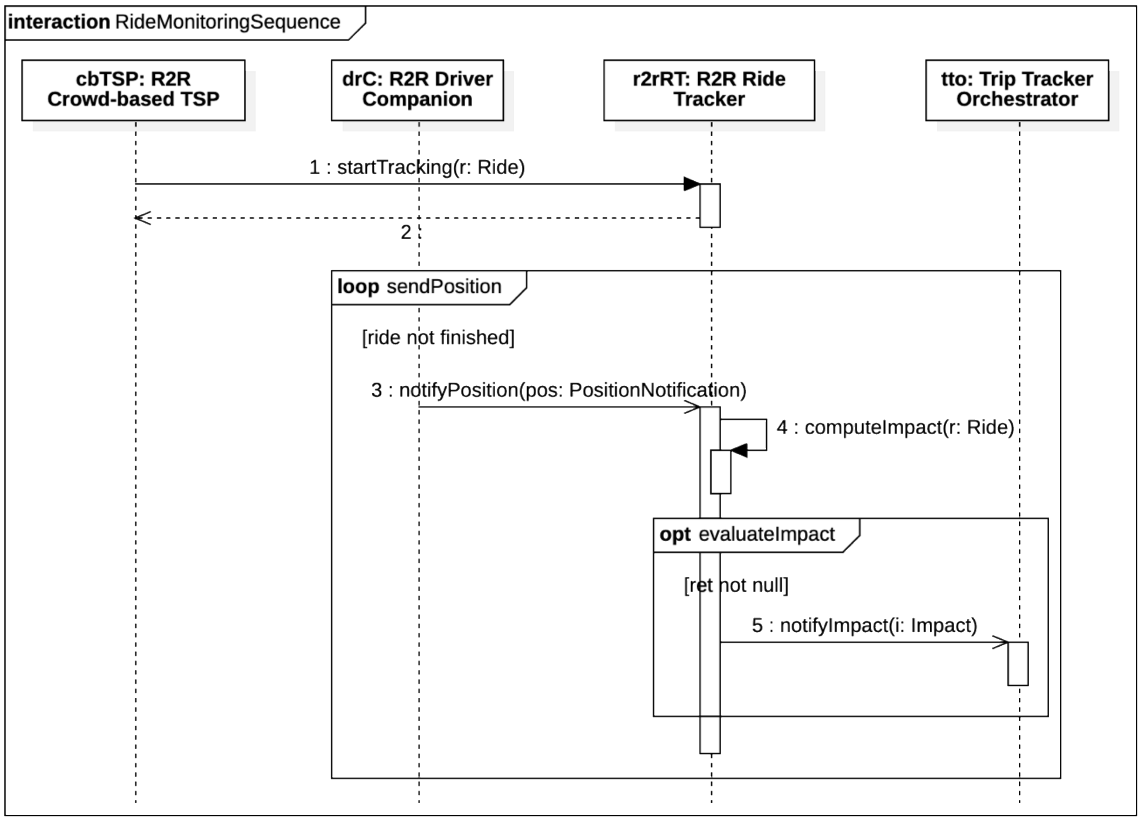
interaction RideMonitoringSequence
cbTSP: R2R Crowd-based TSP
drC: R2R Driver Companion
r2rRT: R2R Ride Tracker
tto: Trip Tracker Orchestrator
1 : startTracking(r: Ride)
2
loop sendPosition
[ride not finished]
3 : notifyPosition(pos: PositionNotification)
4 : computeImpact(r: Ride)
opt evaluateImpact
[ret not null]
5 : notifyImpact(i: Impact)

Figure 13 – Sequence Diagram capturing the interaction concerning the computation and notification of impacts.

7.4. Deployment

Figure 14 depicts, through a UML Deployment Diagram, how the modules identified in Section 7.2 can be distributed across the nodes of network, to identify which functions should be offered as services, and which functions should instead be realized by modules local to existing components.

In particular, the figure shows that The R2Rapp and R2RDrApp (i.e., the Personal Application part of the Travel Companion and the Driver Companion) should be deployed on the same mobile device (assuming the user is both a Passenger and a Driver, so they use both applications). It is still left open the possibility that these two applications are even more tightly integrated between them in a single application, though currently this does not seem to be the preferred solution.

Contract No. 881825

The *ActivityMonitors* (on the Passenger and on the Driver side) must be tightly integrated with the respective applications; they cannot be offered as services, since they need to directly monitor the behavior of the user.

All other components, instead (OfferEnricherAndRanker, OfferCategorizer, LearningAndClustering, etc.), are envisaged to be offered as services (they reside on a separate node than R2Rapp and R2RDrApp).

Finally, the R2RCbTSP will also provide its functions through a separate endpoint that is accessible not only by the user applications, but also by the services (Shopping Orchestrator, Trip Tracker Orchestrator, etc.) that are part of the S2R ecosystem.

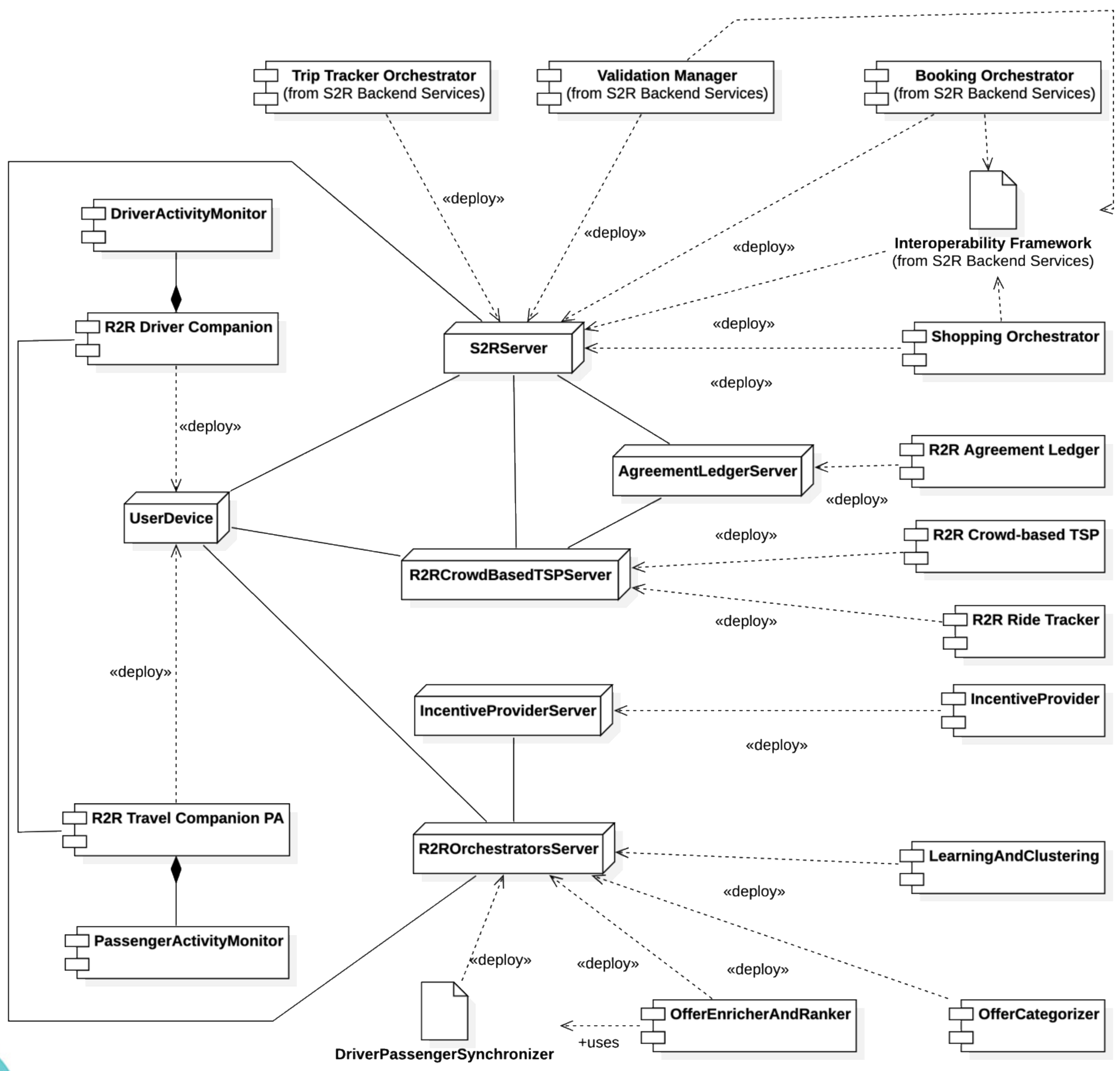

Figure 14 – Deployment of R2R and S2R components across servers.

7.5. Requirement coverage by components

The traceability matrix shown in Table 32 details, for each requirement identified in Section 6.4, which components contribute to fulfil it.

Table 32: The coverage of requirements by components.

		R2R Crowd-based TSP	DriverActivityMonitor	R2R Driver Companion	LearningAndClustering	R2R Travel Companion PA	PassengerActivityMonitor	OfferCategorizer	OfferEnricherAndRanker	IncentiveProvider	DriverPassengerSynchronizer	R2R Agreement Ledger
R2RDrApp	Req1			X								
	Req2			X								
	Req3	X		X								
	Req4	X		X								
	Req5			X								
	Req6	X		X								
	Req7	X		X								
	Req8			X								
	Req9		X	X								
	Req10			X	X							
	Req11			X								
	Req12			X								
	Req13			X								
	Req14	X		X								
	Req15			X								
	Req16	X		X								
	Req17			X								
R2Rapp	Req18					X						
	Req19					X						
	Req20					X						
	Req21					X						
	Req22					X						
	Req23					X		X	X	X	X	
	Req24					X	X					
	Req25				X	X						
	Req26					X				X		
	Req27					X						
	Req28					X						
	Req29					X						
	Req30					X						
	Req31					X						
R2R ChT	Req32	X		X		X						X
	Req33	X										

	Req34	X										X
	Req35	X										X
	Req36	X										X
	Req37											X

Contract No. 881825

8. FEASIBILITY OF INTEGRATION WITH EXISTING COMPONENTS

This section discusses the feasibility of integrating the functions identified in Section 6 and Section 7 into existing modules developed within other projects, and in particular by other S2R IP4 projects.

We focus on three artifacts:

- The TC developed in the S2R CFM ATTRACkTIVE project and deployed in the COHESIVE CFM project.
- The framework developed within the My-TRAC[5] OC project.
- The ride-sharing service developed within the H2020 SocialCar project.

For each artifact, an overview is provided, and some considerations are drawn concerning the feasibility of integrating R2R modules into them, or of using them as the basis for the R2R developments.

In addition, at the end of this chapter a final section highlights potential issues that have been raised during discussions with CFM project members, which can hamper the integration of some of the functions and mechanisms described in Section 6 into the S2R ecosystem.

8.1. The ATTRACkTIVE Travel Companion

The general structure of the TC developed within the ATTRACkTIVE project is comprised of a framework built for Android applications that can accept and incorporate different modules for independent features. The framework was discussed in several meetings held with partners of the ATTRACkTIVE and COHESIVE projects. The rest of this section provides an overview of the main features (from the point of view of R2R needs) of the TC, as derived from the documentation made available by the CFM projects and from the aforementioned discussions. Figure 15, which is discussed in the following, shows the positioning of the ATTRACkTIVE TC within the S2R ecosystem.

[5] http://www.my-trac.eu

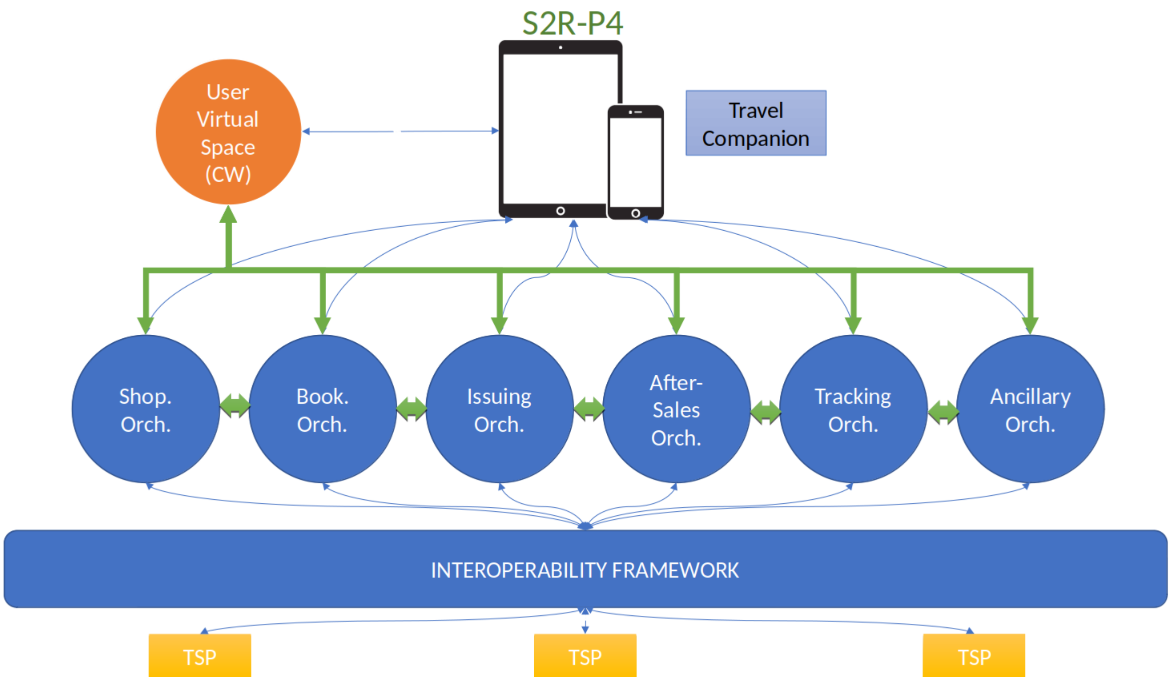

Figure 15 – Positioning of the ATTRACkTIVE TC within the S2R context (courtesy of COHESIVE)

The main goal of the structure of the TC is to keep the mobile front-end as thin as possible, moving all the business logic to "orchestrators" that are part of the S2R ecosystem. Additional feature modules can be included in the mobile front-end through the so-called Navigator Library (Navigator lib for short) that has been created specifically for this purpose: the library allows for not only the incorporation of new modules into the TC, but also the addition of UI components that directly interact with the user.

HACON EXPLANATION

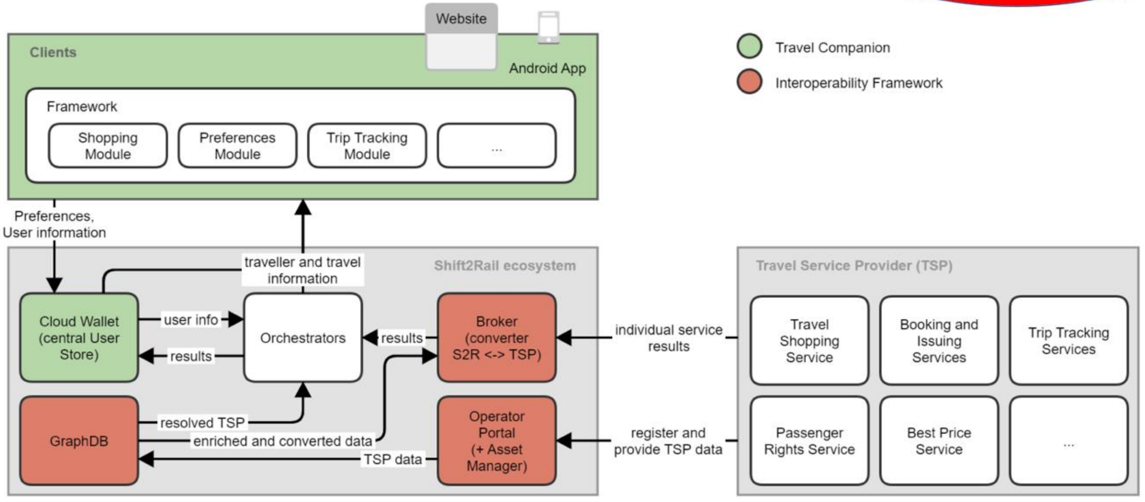
Website
Clients
Android App
Framework
Shopping Module
Preferences Module
Trip Tracking Module
...
Preferences, User information
traveller and travel information
Shift2Rail ecosystem
Cloud Wallet (central User Store)
user info
results
Orchestrators
results
Broker (converter S2R <-> TSP)
GraphDB
resolved TSP
enriched and converted data
TSP data
Operator Portal (+ Asset Manager)
individual service results
register and provide TSP data
Travel Service Provider (TSP)
Travel Shopping Service
Booking and Issuing Services
Trip Tracking Services
Passenger Rights Service
Best Price Service
...
Travel Companion
Interoperability Framework

Figure 16 - Overview of the TC components (courtesy of COHESIVE)

These new modules are meant to be standalone components which provide an interface to call specific services in the S2R ecosystem (see Figure 16). Other components inside the TC can interact with the added feature module by using supplied interfaces, but this communication must be one-sided: the feature module knows nothing about other components of the framework it is included into, nor it has access to their functions as it is not able to use the functional interfaces of the core module of the TC. Moreover, a new feature module should contain the bare minimum amount of business logic required for it to work and it should rely on services and orchestrators belonging to the S2R ecosystem for any kind of required operations computations.

The feature module can be designed to make use of push notifications coming from the S2R ecosystem. The module has to provide an entry point to be called whenever a notification is sent to the TC application in order to activate the module itself; a process then should be instantiated to check whether the notification is meant for this module or not based on its own internal parsing mechanisms that would pick up requests directed to it.

Some of the functions to be developed within the R2R project concern the monitoring and handling of user preferences. In particular, the learning mechanisms to be developed within R2R require the collection of information concerning the interactions of the user with the TC. This can be achieved thanks in part to the so-called Cloud Wallet (or User's virtual space) of the TC, which stores and, to a certain extent, makes available to external components, user information such as their preferences. More precisely, the collection of information such as selected options and performed actions can be achieved by accessing the tracker interfaces available in the Navigator lib, which is able to generate tracking events describing which screens are shown, or which actions are triggered by the user. This information can then be processed by the app and sent to the Cloud Wallet, which could then be accessed by specialized analytics frameworks.

As mentioned in the description above, the key technical element of the ATTRACkTIVE TC is the Navigator lib. Given its importance, the next section provides some additional details about the technology (documentation for the Navigator lib is not publicly available, but it has been confidentially provided by COHESIVE partners to the R2R project).

8.1.1. Navigator Library

The Navigator lib is a library bundle that should be integrated in any module R2R develops for the mobile side of the TC.

The library provides functional interfaces to include new modules into the TC and allow them to access some shared features of the Navigator lib. This is achieved by specifying so-called `Intents` available via the `HafasApp`, which is an adapter class for accessing the library's core features. Once the `Intent` is specified, it can be used to start activities implemented in submodules of the bundle, for example a planner in a journey planning module, or the mobility map in a map module. Events and push notification functions can be accessed using this strategy as well as backup and shutdown interfaces to save current session data via the library.

Moreover, the Navigator lib provides a tracker interface that allows third-party libraries to track active screens and user's behaviour. `trackEvent` and `trackScreen` are access points of the `ExternalTracker` interface that can be implemented to collect logs, which can be then forwarded to the TC Wallet, to secondary learning orchestrators or specialized analytics frameworks.

Lastly, R2R modules can use the Navigator lib also to add views for user navigation within the application, by implementing the `NavigationMenuProvider` interface and injecting it back to the Navigator lib during the initialization phase. It provides a mean to add UI components that the user of the application can interact with, starting from in-app navigation components (a drawer by default) that should be attached to the shared navigation drawer.

The `HafasContext` interface provides access to library-specific UI operations.

8.2. My-TRAC

The My-TRAC S2R OC project developed its own Travel Companion, and a set of modules to improve the user experience. As part of the collaboration with the My-TRAC project, the R2R partners received some documentation providing an overview of the features developed within it, and, in particular, its backend services. The following paragraphs will focus mainly on the My-TRAC architecture and how it can be used in R2R.

The My-TRAC TC is a thin application, similarly to the Personal Application of the ATTRACkTIVE TC, in which the business logic is handled by server-side modules (see Figure 17). It relies on a proprietary CIGO! platform for the implementation of algorithms and data requirements used.

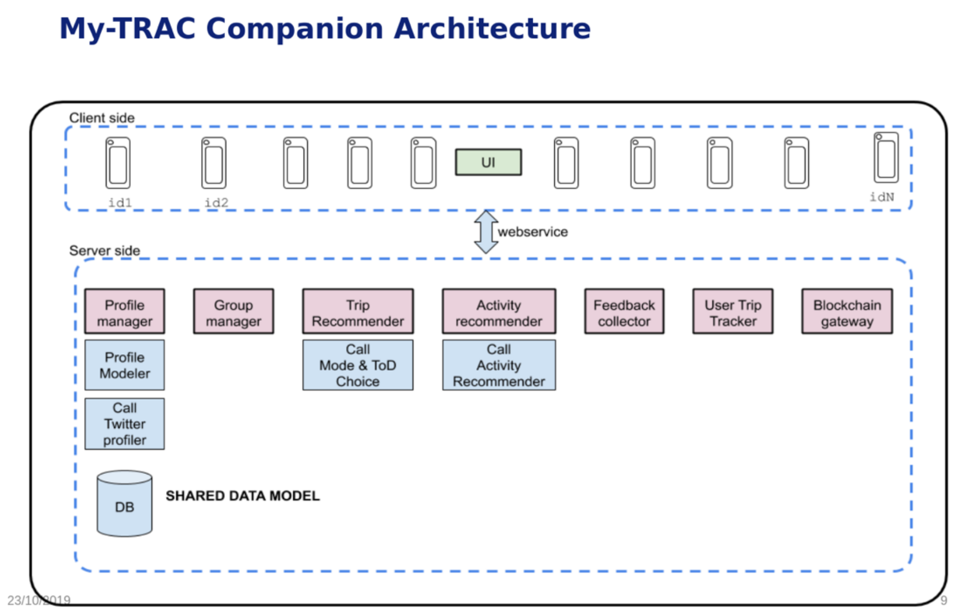

Figure 17 - Overview of the My-TRAC TC (courtesy of My-TRAC)

What sets the My-TRAC system apart from the ATTRACkTIVE one is that the CIGO! Platform can act as a communication bus between connected applications as well.

Overall architecture

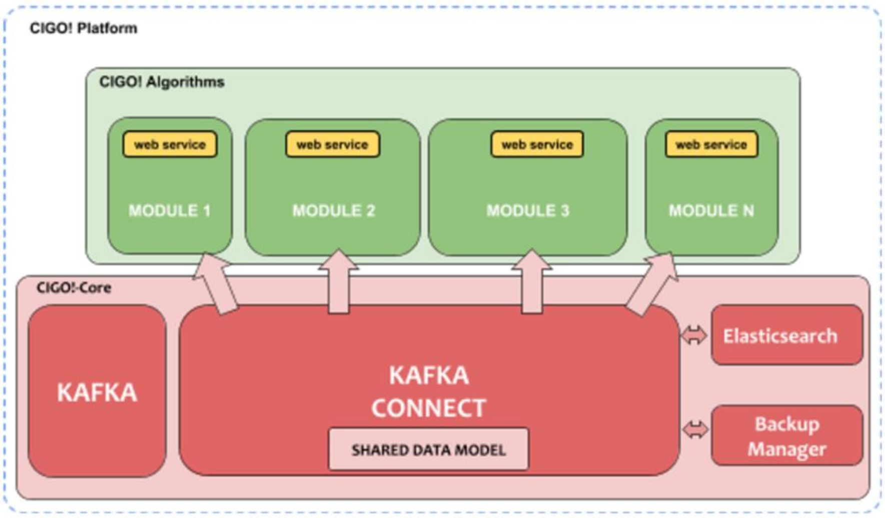

Figure 18 - Overview of the CIGO! platform (courtesy of My-TRAC)

Figure 18 illustrates the architecture of the CIGO! platform. It is a platform for event sourcing that is based on the Confluent stack[6] and is divided into two separate logical parts: one is dedicated to algorithms, and the other to core functions.

The latter is responsible of keeping a persistent source of truth for all events that conform to the data model defined in the My-TRAC project,[7] and provides and interface to publish or poll such events classified by their topics, thus allowing asynchronous communication between different components of the system. Additional modules can be included in this section if required.

The former, on the other hand, consists of a set of modules that observe event streams running through the platform and produce new events conforming to the specified data models that can be used for profiling, routing and user preferences.

8.3. SocialCar

The architecture of the system developed within the H2020 SocialCar project (see Figure 19 for an overview) mixes the Data-flow and Data-centered paradigms. It has

[6] https://www.confluent.io/
[7] https://github.com/My-TRAC/data-model

a slim middleware based on a RESTful Web Server that provides access to all services and modules accessible by the SocialCar Travel Companion application. This allows for the possibility of adding custom modules providing additional features as required by R2R. Similarly to My-TRAC, the backend of the SocialCar system is divided into two parts: one for algorithmic solutions (route planning, matching, reputation assessment, etc.), and the other for service modules (user management and payment).

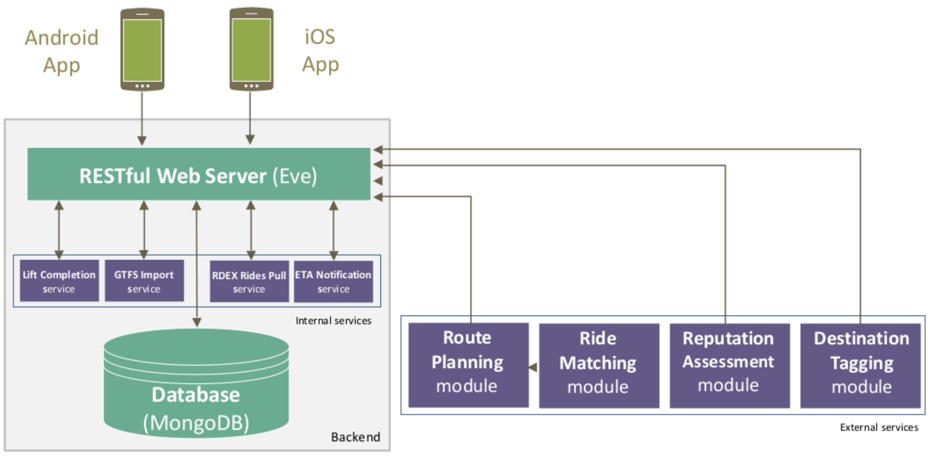

Figure 19 - Overview of the architecture of the SocialCar system (from SocialCar's documentation)

Many of these modules rely on querying the main database for static data, rather than accessing a service or an API gateway to query TSPs directly: for instance, car pooling data has to be loaded in the database daily. The RDEX Rides Pulling service shown in Figure 19 and spotlighted in Figure 20 does just that: it gathers data provided by external carpool providers and fetches rides, then it parses and imports them into the database using the chosen standard (Ridesharing Data Exchange, or RDEX[8]) to facilitate requests among different operators.

[8] http://rdex.org

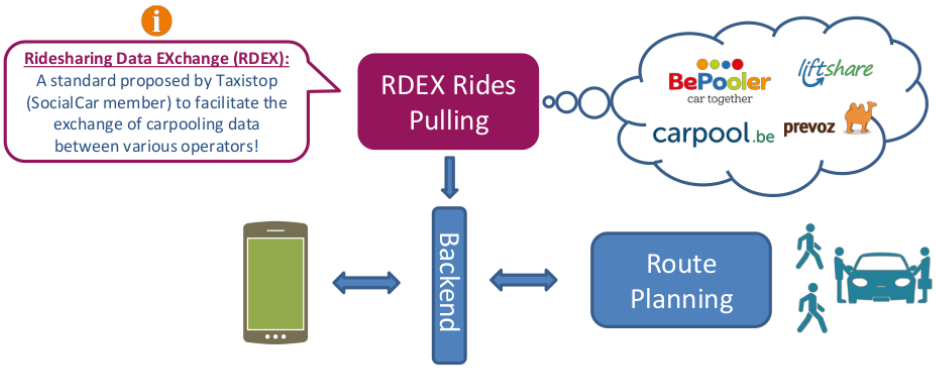
Ridesharing Data EXchange (RDEX):
A standard proposed by Taxistop (SocialCar member) to facilitate the exchange of carpooling data between various operators!
RDEX Rides Pulling
BePooler car together
liftshare
carpool.be
prevoz
Backend
Route Planning

Figure 20 - Overview of the ride pulling mechanisms (from SocialCar's documentation)

Some of the modules developed within the SocialCar project could be used by the R2R project for building its own Crowd-Based TSP service: the Ride Matching component, for instance, works tightly with the Route Planning module and is used to establish an agreement between the rider and the driver. It provides key features such as dynamic traveler assignment to carpoolers and seat request distribution on carpooling vehicles. Used in conjunction with push notification services provided by SocialCar, it allows secondary services such as Driver Expected Time of Arrival (ETA) notification to be implemented.

Closely tied to the previous component there are the Reputation Assessment module, which generates clusters of users based on their travel preferences and calculates/assigns a reputation score to a user based on feedback, and the Destination tagging module, which analyzes user trajectories collecting data from GPS recording and extracts common sub-trajectories among users (travelling peers' suggestion) identifying the nature of these trajectories (home-work, home-leisure). Both these aspects can be included into the Preference module that R2R is going to develop.

The Travel Companion application, on the other hand, will only see and access the REST API endpoints provided by the Web Server, and internal components of the backend can query secondary modules for custom features or complex processing, making it easier to upgrade an existing feature module (for example, the one for Route planning), and enrich it with additional functions developed by R2R (such as offer ranking).

Additionally, there is a live reporting section in the SocialCar Travel Companion application to share traffic information, accidents and other events which can be done either by users or devices. The application itself has already been developed having a separate kind of user in mind, the Driver, making it easier to adapt it into a fully-capable standalone application that would serve as a Crowd-based TSP as well.

8.4. Evaluation

All three surveyed systems adopt a “thin-client” approach, where the user interface (i.e., the Personal Application side of the TC) relies heavily on the backend for the handling of the business logic: most of the logic should be handled by orchestrators (in the S2R parlance) and by additional service providers that are external to the TC application itself.

Concerning the **ATTRACkTIVE TC**, we identified two ways in which new functions can be added: either by creating a fully operational AAR library, to be integrated by the TC maintainers, that would encapsulate all required procedures, or via a new orchestrator, that would live in the R2R ecosystem and that would expose an interface the TC could access.

Directly accessing and modifying the TC codebase, on the other hand, does not seem to be an option: any new features developed by the R2R project should be provided as services to be invoked as-is by the ATTRACkTIVE TC. It should be noted that in both cases previously described most of the business logic is expected to remain on the cloud, thus limiting the impact of new modules on the TC Personal Application itself; as such, these AAR modules and services should be as self-contained as possible, limiting interactions with external components to the bare minimum.

The Navigator library described in Section 8.1.1 can be used to add feature modules that will provide bridges to backend components, such as remote orchestrators and other cloud-based services with non-trivial logic. Moreover, it should be these components that oversee activating or deactivating required functionalities (e.g., position tracking) within the feature modules in a reactive manner. The modules will be using the Navigator lib `Intent` to access device settings (WiFi/GPS localization), but using a push notification from the corresponding orchestrator seems to be the best option to begin the required process.

Since push notifications are intercepted and handled first by the core modules of the ATTRACkTIVE TC itself, feature modules are required to have an access point for them and a filtering mechanism to set apart those meant for the module from the others.

The Navigator lib also offers features to monitor the interaction of users with existing modules. However, it seems that not all modules that are part of the ATTRACkTIVE TC rely on the Navigator lib for their implementation (e.g., the Preference module). Tracking screen and application usage via the Navigator lib functional interfaces is not possible for these components and requires a different approach that is yet to be specified, which unfortunately could limit the development of a learning module or service orchestrator.

My-TRAC on the other hand, offers R2R the possibility of integrating CIGO! platform into the ecosystem. Having an event sourcing platform in place is extremely useful for many different types of activities like, for instance, trip tracking: lots of events generated can be gathered and parsed by a separate component and then fed back to orchestrators in the ecosystem to perform evaluations such as delays or estimations.
Additionally, CIGO! can be used for learning purposes as well: the TC can be set to send information to R2R about UI interactions, preferences and activities that would be separated by purpose or topic in the platform and dispatched to the required components.

For what concerns SocialCar, there are multiple ways in which the artifacts developed in the project can be used by R2R.

The first one concerns the R2R Driver application (R2RDrApp), which could make use of some of the features described in the previous section and of the UI already developed by SocialCar for this purpose. More precisely, the latter can be adapted into becoming the hub for Crowd-based TSP functions, such as those for creating new rides or managing existing ones, accepting and validating passengers, notifying delays or rescheduling the rides and so on.

The second one concerns the possibility of deploying modules developed for the SocialCar backend within the R2R ecosystem. This means that existing modules such as the Ride Matching one can be integrated directly to be used as an orchestrator or a service in the ecosystem. Unlike other projects, SocialCar already supports multimodal solutions that include carpooling and carsharing, thus the integration should not require significant changes on the application side. In addition, since the codebase is opensource, modules like Destination Tagging and Reputation Assessment can be enriched directly with additional functions, or they can be extended to be used as a base for new feature modules such as the Learning one.

8.5. Possible Integration Issues with Shift2Rail Ecosystem

This section highlights some potential issues that could hamper the integration of the R2R functions and components into the existing S2R ecosystem. These issues have been highlighted during discussions with CFM project members and through the feedback received from them.

Contract No. 881825

One potential issue concerns the type of interaction that the Booking Orchestrator (see Figure 15) expects to have with TSPs when the booking of a selected offer occurs. In particular, currently the Booking Orchestrator expects that, when it contacts a TSP to confirm an Itinerary Offer Item handled by the TSP, the TSP is able to immediately determine whether the offer can be booked or not. Contrast this with use case UC11, in which the booking of a shared ride requires confirmation from the Driver, which necessarily cannot be done synchronously, at the same time of the Passenger's request. Notwithstanding alternative mechanisms where a Driver allows the Crowd-based TSP to automatically confirm a booking in their stead without explicit confirmation, given the peculiar nature of shared rides (which are offered by private citizens who are not employees of the TSP), it seems important that Drivers are allowed to review the booking requests for shared rides they offer. This is indeed the standard approach in existing services such as BlaBlaCar (www.blablacar.com). The realization in practice of the mechanisms described in UC11, however, does not seem to be possible through either of the two approaches (realization of an ARR library or of a new orchestrator) described in Section 8.4 to integrate features in the S2R ecosystem, and in particular to extend the features of the existing S2R TC; indeed, a modification of the interaction of the TC with the Booking Orchestrator appears to be necessary. An operative decision regarding how to tackle this issue is outside of the scope of the present deliverable; the R2R project will further investigate this issue in the context of WP3.

A second issue is the integration and relationship between Travel Companion (i.e., the front-end application on the Passenger side) and Driver Companion (i.e., the front-end application on the Driver side). As described in Section 7.2, they could be developed/deployed as separate applications, or as a single front-end capable of satisfying both the needs of Passengers and those of Drivers. A clear preference has been expressed by CFM projects to keep the two applications separate, though communicating with one another. Hence, the R2R project plans to follow this direction, though the situation will be monitored and re-evaluated in the future within WP3 to ensure that a smooth connection between the two applications can indeed be established.

Still, even the realization of a separate application for the Driver Companion would entail that some modifications should be applied to the existing S2R TC (for example, to allow the Passenger to send the information concerning the shared ride to the Driver Companion). Modification of the existing S2R TC cannot be achieved by the R2R project by itself, but it requires a direct intervention on the part of the S2R TC maintainers within the CFM projects.

9. CONCLUSIONS

This deliverable lays the groundwork for the development of the modules implementing the techniques and processes defined in the R2R project.

At a macro level, the functions to be developed in the R2R project are the following:

- Mechanisms to categorize offers according to their characteristics (e.g., eco-friendliness).
- Mechanisms to provide incentives to users (Passengers and Drivers) to adopt virtuous behaviors (e.g., in terms of sustainability).
- Mechanisms to automatically learn user preferences and use them to provide travelers with offers that better fit their tastes.
- A full-fledged TSP (providing features such as trip planning, booking, entitlement and token issuing, trip tracking) tailored to the handling of shared rides, which can be integrated in the S2R ecosystem.
- A decentralized, reliable and secure storage (the so-called Agreement Ledger) to enable trustworthy auditability of important events during a shared ride and the automatic resolution of disputes, saving time and costs.

Starting from a set of use cases that describe how external actors are supposed to use the functions provided by R2R, and how the R2R system behaves in response to user behaviour, the deliverable identifies an initial set of requirements that the R2R system should fulfil. Then, it defines a set of components – and interfaces thereof – that will implement the aforementioned requirements, and it describes their interactions to show how the components are supposed to work together to fulfil the requirements. To prepare for the realization of the identified components, the deliverable suggests a possible deployment scheme, identifying which functions should be provided as new services, and which ones should be integrated as extensions of an existing TC. Finally, the outcomes of some companion projects (ATTRACkTIVE, My-TRAC, SocialCar) have been analyzed, to study the feasibility of, on one side, implementing the new R2R modules based on existing technologies developed by those projects and, on the other side, integrating these new modules in the S2R ecosystem.

The present deliverable has described the current status concerning the approach to be followed to integrate the R2R functions into the S2R ecosystem. Though the architecture presented in Chapter 7 reflects the approach that has been identified so far as the best, it is still possible that the structure will be revised in the future, during the developments carried out in WP3.

Contract No. 881825

Contract No. 881825

10. REFERENCES

[1] RIDE2RAIL Consortium, “D2.3: First set of Requirements and Specification for Complementary Travel Expert Services,” 2020.

[2] RIDE2RAIL Consortium, “D2.2: State-of-the-art of ride-sharing in target EU countries,” 2020.

[3] RIDE2RAIL Consortium, “D2.1: First conceptualization of choice criteria and incentives,” 2020.

[4] MaaSive Consortium, “D11.1 - CREL Glossary,” 2020.

[5] RIDE2RAIL Consortium, “D2.4: Final conceptualization of choice criteria and incentives,” 2020.

[6] RIDE2RAIL Consortium, “D2.5: Recommendations and criteria for a successful ride-sharing in the IP4 ecosystem,” 2020.

11. ANNEX – COMPLETE UML MODEL

The complete UML model can be found in the RIDE2RAIL_system_model.mdj[9] file that accompanies this deliverable. A PDF version of the model is also made available.

[9] The model was created through the StarUML tool (a free version of the tool, with which the model can be read, is available from http://staruml.io).

Contract No. 881825