QUANTITATIVE ESTIMATE OF SERVICE QUALITY FACTORS

Deliverable D3.1

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	01/05/2021
Duration (in months)	30
Deliverable number	D3.1
Call Identifier	S2R-OC-IP4-01-2019
GRANT Agreement no	881825
Due date of the Deliverable	31/03/2021
Actual submission date	07/10/2021
Responsible/Author	Cristian Consonni (EUT)
Dissemination level	PU
Work package	WP3
Main editor	Cristian Consonni (EUT)
Reviewer(s)	Fabio Cartolano (FIT), Matteo G. Rossi (POLIMI)
Status of document (draft/issued)	issued

Reviewed: yes

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
0.1	14/06/2021	First complete draft
0.2	29/09/2021	Final draft for review
0.3	04/10/2021	Final editing after the reviews

REPORT CONTRIBUTORS		
Name	Beneficiary Short Name	Details of contribution
Cristian Consonni	EUT	Section 2, 3.1, 3.2, 3.3, 4.2, 4.4, 4.5, 4.6, 5.5, 6
Alex Martinez Miguel	EUT	Section 4.4, 5.1, 5.5, 5.6
Luca Piras	EUT	Section 4.4, 5.1, 5.5, 5.6
Lodovico Boratto	EUT	Section 2, 4.6, 5.5, 5.6, 6
Ľuboš Buzna	UNIZA	Section 3.3, 4.2, 4.4, 4.5, 5.3, 5.5
Milan Straka	UNIZA	Section 4.4, 5.5
Yannick Cornet	UNIZA	Section 4.5
Tatiana Kováčiková	UNIZA	Section 4.5
Alireza Javadian Sabet	POLIMI	Section 4.2, 4.5
Georgia Ayfantopoulou	CERTH	Section 5.5
Anna Kortsari	CERTH	Section 5.5
Lambros Mitropoulos	CERTH	Section 4.5, 5.5
Mario Scrocca	CEF	Section 3.3, 4.3, 4.4, 4.5, 4.6, 5.2, 5.4

Contract No. 881825

Disclaimer

Table of contents

List of Figures

List of Tables

Contract No. 881825

1. EXECUTIVE SUMMARY

In this Deliverable we present the architecture and implementation of the Offer Categorizer (OC) developed in the scope of the RIDE2RAIL (R2R) project. RIDE2RAIL[1] (H2020-Shift2Rail-881825) has the overall objective of developing an innovative framework for intelligent multimodal mobility, by facilitating the efficient combination of flexible and crowdsourced transport services, such as ridesharing, with scheduled transport. This framework will be integrated natively into existing collective and on-demand transport services, connecting and reinforcing the mobility offer with ride-sharing services especially in rural and low-demand areas, facilitating access to high-capacity services (rail, bus, and other public transport services) thanks to easy-to-use multi-modal and integrated travel planning, booking, ticketing, and payment features. The R2R framework will integrate and harmonize real-time and diverse information about rail, public transport, and shared mobility in a social ecosystem, allowing users to compare and choose between multiple travel offers classified by a set of criteria including environmental impact, travel time, travel cost, and comfort.

This deliverable describes the results of Task 3.1 (T3.1), part of Workpackage 3 (WP3), where we introduce a novel system to enrich and classify multimodal travel solutions proposed through the Travel Companion (TC). We built upon the conceptualization provided in D2.4 - “Final Conceptualization of Choice Criteria and Incentives” and we developed a state-of-the-art Offer Categorizer to enable the description of offers along 11 different categories, evaluating each trip offer along several dimensions such as comfort, environmental friendliness, and health impact. This classification enables the implementation of a matching and ranking algorithm, that also learns the user's preferences over time based on their travel choices. Users will thus benefit from the enrichment of travel offers provided by the IP4 ecosystem and more personalized travel solutions over time.
In practical terms, the Offer Categorizer receives a set of travel offers – i.e., the proposed travel solution for a mobility request made by an IP4 user – in TRIAS format. TRIAS, which stands for Travellor Realtime Information and Advisory Standard, is a transport standard that has been developed in the scope of the research and standardisation project for public transport "IP-KOM-ÖV" and was then introduced in 2014 as a standardized specification by the Verband Deutscher Verkehrsunternehmen. TRIAS offers a wide-range list of functionalities, including station and location search, real time departures, navigation, ticket price calculation, and malfunction reporting.
The first step of the processing performed by the OC is to extract from the TRIAS format the characteristics of each travel offer that are relevant for its classification. Then, a set of Feature Collectors (FCs) compares the offer along a given peculiar characteristic – or

[1] https://ride2rail.eu/

Determinant Factor (DF) – that forms the foundation for the computation of a score for a given category.
Each software component belonging to the OC has been developed as a microservice, to facilitate its deployment in a production environment. We have used industry-standard practices for the development and all the code that we have developed in the scope of this task is already publicly available on GitHub under the Ride2Rail organization.[2]
This Deliverable is intended to be a companion to the software documentation available on GitHub under the RIDE2RAIL organization at https://github.com/Ride2Rail, where it is possible to find the full source code for every component of the OC. In this Deliverable, it is provided the full source code and it is described the context and challenges that we have faced during the development of the OC.
This deliverable is structured as follows: Section 3 provides the background and context for the Offer Categorizer, which objectives and requirements we had to reach and which constraints we faced. We also provide a description of the backstory related to the development of the OC. Section 4 describes the design of the OC, first by providing a general description of the main intuition at the basis of the OC and then describing in detail the design decision taken, accompanied by their rationale. Section 5 recounts the development process of the OC, by introducing in detail the software technologies utilized, the Algorithms. Section 6 provides the conclusions.

[2] https://github.com/Ride2Rail/

Abbreviations and acronyms

CFM	Calls for Members
DF	Determinant Factor
DL	Dissemination and exploitation leader
DoA	Description of the Action
EL	Ethical leader
EU	European Union
FC	Feature Collector
FS	Financial Statement
GA	Grant Agreement
H2020	Horizon 2020
IP4	Innovation Programme 4
OC	Offer Categorizer
PC	Project coordinator
PM	Project manager
PMO	Project Management Office
PMT	Project Management Team
PO	Project Officer
QAC	Quality Assurance Committee
R2R	Ride2Rail
ROD	Rank order distribution
S2R JU	Shift2Rail Joint Undertaking
TRIAS	Travellor Realtime Information and Advisory Standard
TL	Technical leader
TSP	Transport Service Provider
WP	Work Package
WPL	Work package leader

Contract No. 881825

2. OBJECTIVE

The present document constitutes the Deliverable D3.1 “Quantitative Estimate of Service Quality Factors” in the framework of the WP3, Task 3.1 - “Development of algorithms to propose quantitative estimates”. We report here the description of the task for clarity.

Task 3.1 Development of algorithms to propose quantitative estimates
This task will exploit the knowledge acquired in WP2 to automatically classify travel offers, i.e., to assign to each offer one or more categories. The classification will be based on different characteristics that describe an offer (considering for instance environmental or comfort aspects). A benchmarking of multiple methodologies (e.g., unsupervised and supervised algorithms) will be undertaken to find out the one that provides the best classification. The classified offers will be then enriched through a quantitative estimate of different service quality factors (e.g., travel time, comfort, reliability, environmental friendliness). Multiple sources of data will be considered in the data enrichment in order to allow an accurate characterization of the proposed travel offers. According to external source of data to be queried, different approaches such as API-based or data crawling-based approaches may be adopted. The classified and enriched offers will feed task 3.2.

The output of this deliverable contributes as well to WP3 tasks T3.2-T3.4 of the RIDE2RAIL project (S2R-OC-IP4-01-2019).

Contract No. 881825

3. CONCEPTUALIZATION OF THE OFFER CATEGORIZER

3.1.Summary of requirements

This section summarizes the offer categorizer (OC) description as presented in the grant agreement (RIDE2RAIL, Grant agreement number 881825, 2020). The module aims at automatically classifying the travel offers computed by the travel service provider (TSP). Precisely, the OC should be able to assign to each travel offer one or more categories according to the results of the work done Work Package (WP) 2, especially (RIDE2RAIL, Deliverable D2.4 - Final Conceptualization of Choice Criteria and Incentives, 2020). To do so, multiple benchmark approaches such as supervised and unsupervised algorithms have been explored to find the most promising approach. The assigned categories should be enriched through quantitative estimation of various service quality factors such as comfort, environmental friendliness, etc. The enrichment step incorporates multiple data sources through API-based or data crawling-based approaches for accurate categorization of the travel offers. Finally, the outputs of the OC, i.e., the travel offers enriched by the appropriate categories, will be used by Task 3.2 - ”Algorithms for optimal synchronisation of shared-mobility and mass transit”.

3.2. Limitations and constraints

This section describes the general approach we have considered for designing the OC, with its requirements and limitations. Given the constraint posed by the project, we decided to opt for a top-down approach, where we used the knowledge gathered in D2.4 to define a set of valuable features to be extracted for each trip to be categorized.

When designing a categorization system and, more generally for any data-driven project, two approaches are possible, labelled “bottom-up” and “top-down,” respectively. The nomenclature refers to a visual representation of the process of information extraction from data as a pyramid - also known as Data Science’s “Wisdom Pyramid” (Figure 1) - which puts raw data at the bottom, and more and more sophisticated algorithm of knowledge extraction as each level of the pyramid is climbed.[3]

[3] Towards Data Science. 2021. The Data Science Wisdom Pyramid. [online] Available at: <https://towardsdatascience.com/the-data-science-wisdom-pyramid-8e5d1d85ae82> [Accessed 6 September 2021].

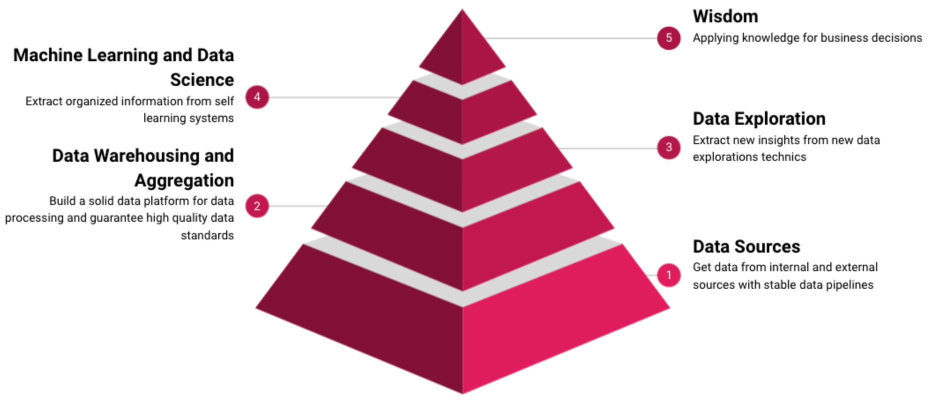

Wisdom
5
Applying knowledge for business decisions
Machine Learning and Data Science
4
Extract organized information from self learning systems
Data Exploration
3
Extract new insights from new data explorations technics
Data Warehousing and Aggregation
2
Build a solid data platform for data processing and guarantee high quality data standards
Data Sources
1
Get data from internal and external sources with stable data pipelines

Figure 1 Data Science's "Wisdom Pyramid".

In the **bottom-up** approach, one starts from having data, and then algorithms are applied to learn from the patterns present in the data to uncover a meaningful classification of data, in our case of travel offers. In this approach, there is a need for an evaluation metric to measure the algorithm's performance used for the classification, in other words, a way to tell how the algorithm is performing on each test case and a dataset. When the expected answer – also known as "ground truth" - is available, we speak of **supervised learning**. For a classification problem, target categories are the ground truth. In this setting, an explicit loss function can be built. The loss function assigns a penalty for each misclassification (error) made by the system. The total loss over a given dataset is the cost of misclassification. A feedback loop tweaks the system's parameters automatically at each interaction to minimize the cost, thus increasing the algorithm's performance. When ground truth information is not available, a different set of approaches, known as **unsupervised learning**, can be applied. In this context, the quality of the results is evaluated using intrinsic metrics that rely on some expected properties of the data. One example of unsupervised learning is clustering: the task of grouping a set of objects so that objects in the same group are more similar to each other than those in other groups. The assumption is that it is possible to define a similarity or distance between the representation of the objects so that similar objects are closed to each other. Bottom-up approaches, both supervised and unsupervised, require a vast amount of data to be effective. These algorithms use an iterative training process that establishes a feedback loop over the parameter of the models.

A way to apply this approach without any data is to use **synthetic data**. Using expert knowledge in the domain of interest, a data scientist can write a program that simulates the process of interest from which samples are picked at random. In this way, a dataset – with or without labels – is available and can be utilized as if they were data coming from a data collection process in the real world.

This approach poses the question of its transferability to the actual final use cases. The system will be able to keep its performance, in terms of accuracy of predictions, or any other metric, only if the synthetic data reflected the same properties of the real-world data. In this sense, the downside of using synthetic data is that it can reduce the design and implementation of a machine learning system to a theoretical exercise, since the parameters set at the end of the training process will depend on the parameters used by the algorithm generating the data. In summary, the effectiveness of this approach in the real world might vary wildly depending on the ability of the generative algorithm to capture patterns occurring in the real world.

A **top-down** approach does not start from data; instead, the characteristics of each category are defined a priori. For example, a trip could be defined as short - i.e., belonging to the short category - when shorter than 10 km. In this approach, the quality of the results of the final algorithm is not expressed in terms of performance against a given metric, but in terms of correctness, i. e., "Does the system do what is expected?".
Analogously to synthetic data, if the definitions provided when designing the classifications rules, the quality of the results may vary wildly depending on the overlap between the definition and reality.

The limitations of any data-driven approach can be mitigated by comparing the results with the data and the data themselves with reality. When implementing a machine learning system (bottom-up) or a rule-based system (top-down), any bias held by the designer or reflected by the data will be present in the final system, hindering its performance in the real world. These biases have consequences on the users, and it is important to acknowledge them and take them into account.
In other words, when designing a solution, the problem should always be kept in sight, and problems occur only in reality. Users should always be put at the centre of any system design: only by observing, interacting, and getting feedback from them, it is possible to design useful and relevant systems.

These approaches are not necessarily mutually exclusive, and we have adopted a hybrid approach for designing the OC. In any case, we require a clear definition of the input and the output of the system, both in terms of semantics and formats. This is especially relevant when the system needs to be integrated withing an existing ecosystem as in the case of R2R.

We faced several challenges during the development of the OC. On one hand, we did not have any ground truth because we did not have any labelled data generated by users. For this reason, we based our approach on the expert knowledge and the results produced by WP2 and we adopted a rule-based top-down approach. On the other hand, we have re-adapted data from other projects to have a realistic set of trips to use as tests when developing the system or we could generate synthetic data. These data sources are described in the following subsections.

Furthermore, since the OC needs to be integrated in the S2R ecosystem we had constraints on the data format that we could use: the input will be in TRIAS format,[4] described more in detail in Section 4.4.1 of this deliverable. For what concerns the output format, after consulting with the CFM partners we decided to use the JSON format.

Finally, the OC categorizer will provide the data for other software modules to be developed within WP3, such as the Offer Matcher and Ranker (Task T3.2), thus requiring a continuous coordination work within the WP and with the CFM partners.

3.3. Summary of inputs from WP2

This section summarises the main relevant contributions from Ride2Rail WP2. In particular, the final conceptualization of offer categories defined in task 2.1, and the requirements and specification for the Offer Categorizer elicited in task 2.3 are presented.

3.3.1. Conceptualization of Offer Categories

The Ride2Rail task 2.1 "*Definition of choice criteria for journey planning*" focused on the conceptualization of choice criteria, in terms of *offer categories*, *user preferences*, and *incentives* for the multimodal door-to-door journey planning. In the first iteration of the task, a first conceptualization was produced considering an analysis of the state-of-the-art and performing an alignment with past and ongoing Shift2Rail IP4 projects. The second iteration of the task, validated and checked the completeness of the proposed first conceptualization by designing, administering and analysing a survey for European travellers.
The final conceptualization of offer categories defined in WP2 supported the design and development of the Offer Categorizer described in this deliverable. This section summarises the main outcomes of Ride2Rail task 2.1 regarding offer categories and introduces their proposed conceptualization. Additional details can be found in Ride2Rail deliverables D2.1 (RIDE2RAIL, Deliverable D2.1 - First Conceptualization of Choice Criteria and Incentives, 2020) and D2.4 (RIDE2RAIL, Deliverable D2.4 - Final Conceptualization of Choice Criteria and Incentives, 2020).

The proposed conceptualization of offer categories has been defined identifying pattern in the related state-of-the-art analysed. It emerged that an offer category is characterised by a specific set of variables (describing the offer) that are the *determinant factors* to determine if an offer belongs to the given offer category.

[4] XSD specification of the TRIAS format https://github.com/VDVde/TRIAS

To refine this intuition, it has been introduced the concept of *Offer Feature*s, each one representing a characteristic of the offer, i.e., the value associated to a variable describing the offer. An *Offer Category* can be seen as a label attached to *Offers* having particular characteristics, i.e., specific *Offer Features*.
Finally, an *Offer Categorizer* is defined as the component that is able to compute the membership of an *Offer* to a given *Offer Category*. The structured set of definitions defined in the final conceptualization can be found in Table 1.
Among the additional details discussed in the task 2.1 deliverables it is important to stress the following remarks on the proposed conceptualization. First, the categorization computed by the *Offer Categorizer* should be independent from the user that is requesting the set of offers. Second, the computation of some offer categories is relative to the set of offers provided as input to the *Offer Categorizer*. Indeed, an offer may belong to a given offer category (e.g., *quick* or *short*) only if considered relative to a set of offers. For this reason, the *Offer Categorizer* expects as input a list of offers.

Table 1 Final Conceptualization of Offer Categories from Ride2Rail deliverable 2.4.

Term	Description
OFFER FEATURE	An *Offer* can be described by a set of objective variables (such as transportation mode, level of CO_2-emission, cost, etc.). The values assigned to the objective variables for a specific *Offer* identify its OFFER FEATURES. For example, *<transportation mode=train>* can be a feature of an Offer. An *Offer Feature* can be computed considering data provided by the TSP (e.g., the price), and/or additional data, e.g., related to the *Trip*(s) associated to the *Offer* in a *Travel Solution* (e.g., length in km), or to the vehicle used in the *Offer* (e.g., CO_2-emission).
OFFER CATEGORY	An **OFFER CATEGORY** identifies a set of *Offer*s having particular shared characteristics. The membership of an *Offer* to a given *Offer Category* is computed considering a set of *Offer Features,* i.e., the *determinant factors* for the *Offer Category*. The membership of an *Offer* to a given *Offer Category* is defined by a ***Category Score* (CS)** in the range of [0,1][1], where 0 means "no membership", and 1 indicates "full membership".
OFFER CATEGORIZER	An **OFFER CATEGORIZER** is a component offering a service to compute the *Category Score*s of an *Offer* with respect to a set of *Offer Categories*. It implements a set of functions that compute the *Category Score*s based on the *Offer Feature*s. The service receives as input the *Offer*s and produces the ranked list of *Category Score*s for each *Offer*.

The final conceptualization of offer categories has been complemented by the definition of a catalogue of offer categories reporting the list of determinant factors that can be considered to compute the membership of an offer to each offer category.
The first version of the catalogue was defined in D2.1 considering the available literature and the outcomes of previous European projects on this topic. First of all, the *types of variables* that can be used to describe a multi-modal travel offer were identified. Then, the *actual*

Contract No. 881825

variables that could be used to define offer categories were elicited to define a set of so-called *low-level offer categories* considering a single determinant factors. Finally, contributions dealing with the identification of *multiple variables* that can be used to define offer categories considering different characteristics of a travel offer were analysed.
As a result, a set of ten offer categories (*Quick, Short, Reliable, Cheap, Door-to-door, Comfortable, Social, Multitasking, Environmentally Friendly*, and *Philanthropic*) each of them combining different determinant factors were identified.

The second and final catalogue of offer categories has been defined in D2.4 considering the results of the survey administered to European traveller passenger. In particular, the collected answers were used to validate and complete the list of offer categories and the associated determinant factors.
As a result, some offer categories were merged (*Social* and *Philanthropic*), new offer categories were defined (*Healthy* and *Panoramic*) and the list of determinant factors for some categories were updated. The final catalogue of offer categories is reported in Table 2. The order of the offer categories in the table reflects the importance associated by the survey participants to the different categories. The most important for users and recommended in the implementation of an *Offer Categorizer* are *Quick*, *Reliable* and *Cheap;* right after it is recommended to consider *Comfortable, Door-to-Door, Environmentally Friendly*, and *Short.* A complete set of offer categories should consider *Multitasking* and *Social. Panoramic* and *Healthy* were added after the survey and therefore it is not possible to compare them with the others.

Table 2 Final Catalogue of Offer Categories from Ride2Rail deliverable D2.4.

Offer Category	Description of Determinant Factors
QUICK	The ***Quick*** category measures how convenient and efficient the solution is in terms of time-related issues, considering the total travel time, the waiting time between legs and the number of stops required. If the solution includes a segment on-road (e.g., bus/car) and real-time data on traffic congestion are available, also these data can be taken into account.
RELIABLE	The ***Reliable*** category concerns the likelihood of delays, traffic congestion, breakdowns or last-minute changes that could affect the travel time and comfort of the trip. Some solutions are inherently variable (e.g., traffic delays when crossing a city at rush hour), while other solutions might offer a small window to change the mode of transport that could cause massive idle times. For this reason, also the frequency of the service for involved solutions should be taken into account. Lastly, the influence of the weather on the trip is taken into account.
CHEAP	The ***Cheap*** category concerns the total price of a trip, the possibility of sharing part of it with others and the ease of payment, giving additional value to solutions that offer an integrated fare system and

Contract No. 881825

	do not require the user to purchase different tickets from different platforms.
COMFORTABLE	The *Comfortable* category concerns objective factors such as the number of interchanges required or the possibility of having a comfortable seat but also covers a set of other elements about the quality of the trip that has to be evaluated through users' feedback. Relevant factors are the cleanliness of the stations and vehicles used and the feeling of personal safety.
DOOR-TO-DOOR	The ***Door-to-door*** category covers the distance of the user's start and endpoint from the beginning and destination locations of the solution provided. It is measured by the amount of walking or driving distance the user has to cover.
ENVIRONMENTALLY FRIENDLY	The ***Environmentally Friendly*** category covers the green aspects of the trip, taking into account at least the amount of CO_2 emissions measured per kilometre/traveller for each mean of transport included in the Offer and considering the distance covered and the number of passengers. If available, additional determinant factors can be considered as the energy consumption, the NOx emissions (nitrogen oxides) and the carbon footprint.
SHORT	The ***Short*** category focuses on minimizing the distance covered.
MULTITASKING	The ***Multitasking*** category concerns the extent to which the user can perform other tasks while travelling. These activities can regard productivity (personal or work), fitness, or enjoyment. This category considers the amount of space available, the presence of silence or business area, as well as whether the internet connection and/or plugs are provided. Lastly, the level of privacy might also influence the extent to which a person can work and could be considered as a determinant factor for this category.
SOCIAL	The ***Social*** category concerns the maximization of the number of people the user will share the trip with and his/her ability to network or socialize based on the context and means used. Moreover, it takes into account solutions that contribute to social causes or involves volunteering or charity activities (e.g., donations).
PANORAMIC	The ***Panoramic*** category promotes solutions passing through beautiful landscapes (like a particular village or a forest) or historical sites. This category also takes into account the usual sightseeing itineraries for tourists to promote solutions passing near monuments or other interesting spots.
HEALTHY	The ***Healthy*** category concerns the involvement of walking and/or cycling in an offer.

A challenging offer category is the *Comfortable* category, given the fact that different works in the literature provide different definitions of "a comfortable offer". For this reason, the administered survey also aimed with a specific question at identifying a shared definition among travellers of determinant factors for the *Comfortable* category. As a result, the cleanliness of the spaces, the comfort of the seats, the feeling of personal safety and the minimization of the number of interchanges were identified as key aspects.

The final set of offer categories implemented in task 3.1, and the selected determinant factors for each category, are discussed in Section 4.2.

Contract No. 881825

3.3.2. Requirements and Specification for the Offer Categorizer

This section summarizes the requirement analysis related to the OC as presented in D.2.6 (RIDE2RAIL, Deliverable D2.6 - Final Set of Requirements and Specification for Complementary Travel Expert Services, 2020).
According to the requirement number 23a (Req23a), "Ride2Rail application (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."
Moreover, the module has been identified as a High-priority module whose function have been covered by two use cases (UC), i.e., UC7 and UC8.

- UC7 is related to generating itinerary offers that possibly include shared rides. This UC details the Passenger's actions while searching for an offer as well as actions related to the R2R systems to build travel offers through the Shift2Rail (S2R) ecosystem.
- UC8 is related to incentivizing Passenger to choose eco-friendlier through Gamification mechanism.

After receiving a mobility request, as show in Figure 2, Shopping Orchestrator builds a set of travel offers and send them to the OfferEnricherAndRanker module. Then, the OfferEnricherAndRanker invokes the OfferCategorizer to enrich the travel offers (i.e., assigning appropriate categories to each travel offer). In the end, the enrichedOffers will be used by the OfferEnricherAndRanker to rank the offers according to the contextual user preferences.

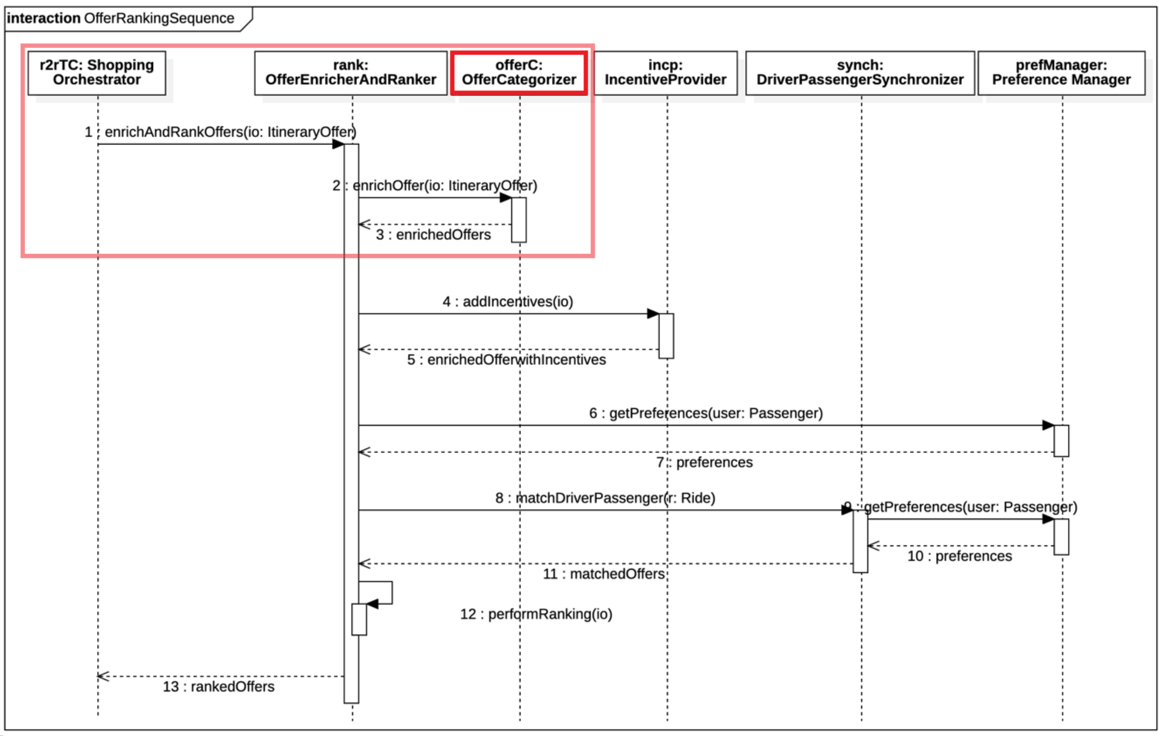

interaction OfferRankingSequence
r2rTC: Shopping Orchestrator
rank: OfferEnricherAndRanker
offerC: OfferCategorizer
incp: IncentiveProvider
synch: DriverPassengerSynchronizer
prefManager: Preference Manager
1 : enrichAndRankOffers(io: ItineraryOffer)
2 : enrichOffer(io: ItineraryOffer)
3 : enrichedOffers
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 2 Sequence Diagram capturing the detail of the interaction concerning the enrichment and ranking of itinerary offers (image from RIDE2RAIL Deliverable 2.6).

4. OFFER CATEGORIZER DESIGN

After a brief description of the main intuition in Section 4.1, Section 4.2 goes through the presentation of the list of determinant factor affecting each category. From this more conceptual description Section 4.3 describes the input data available in IP4 and their structure, together with the structures for storing additional features that are needed for the characterization. Section 4.4 described the data source we used for the development, both synthetic and real, coming from MoTiV, another European project. Section 4.5 describes the assumptions and prerequisites, and methodology for the quantification of offer categories. Finally, Section 4.6 describes in detail the architecture of the Offer Categorizer.

4.1.Intuition

The Offer Categorizier is built to be a flexible, modular system. We base the classification of trip offers on the computation of several determinant factors, which represent relevant feature for the computation of category. After the system receives the input data, I.e. the travel offers in TRIAS format, are received parsed successfully, we need to process the input to compute the features for our classification. This computation is done inside independent **features collectors**, that is services that take a subset of the input data and compute the values for some feature for each travel offers. These values are then aggregated and combined to form the final score for each category, represente by a real number in the range [0, 1].

4.2. Offer categories and Determinant Factors

By reviewing the scientific literature and by evaluating the survey with potential travellers across the Europe, ten offer categories, presented in Table 2, were proposed. By expert design, to quantify each category a list of determinant factors was established. Obtained association between offer categories and determinant factors is presented in Table 3. The following subsections define the meaning of considered determinant factors.

Table 3 The final set of Offer Categories and their Determinant Factors.

Offer Category	List of Determinant Factors
QUICK	Trip duration; Waiting time; Number of stops; Road-network distance/Air-distance; Traffic
RELIABLE	Likelihood of delays; Traffic; Last-minute changes; Time of trip (rush vs non-rush hour); Frequency of service; Weather forecast (day of the trip)
CHEAP	Total price; Can share cost; ticket coverage
COMFORTABLE	Number of interchanges; User feedback; Cleanliness (station/stop/vehicle); Seating quality; Amount of space available, crowdedness; Presence of silence area; Privacy level; Ticket coverage; Ride smoothness; Weather forecast (day of trip)

DOOR-TO-DOOR	Starting point; Ending point; Walking distance
ENVIRONMENTALLY FRIENDLY	CO2 emission/km per traveller
SHORT	Distance covered; Number of stops
MULTITASKING	Amount of space available; Presence of silence area; Presence of business area; Internet connection availability; Plugs/charging points; Privacy level
PANORAMIC	Number of landscapes; Number of historical sites; Number of monuments
HEALTHY	Fraction of the trip by bike/walk; Length of fraction of trip by bike/walk

4.2.1. QUICK

Definitions of Determinant Factors:

- **Trip duration:** overall trip duration including all connections. It is computed by subtracting the scheduled departure from the scheduled arrival [real value; applicable to all modes],
- **Waiting time:** Overall waiting time between trip legs [real value],
- **Time to departure:** waiting time between the moment the user looks for trips and the corresponding departure time of each offer. [real value; applicable to all modes]
- **Number of stops:** number of stops made by a vehicle during the journey [numeric value; applicable to all modes of transport],
- **Road-network distance/Air-distance:** Distance between the starting and ending point done by road/air. In case of multiple legs, the distance is summed over each leg [real value; applicable to all modes],
- **Traffic:** Measurement of real-time traffic (e.g. average speed, density of vehicles) [real value; applicable to trips by roads].

4.2.2. RELIABLE

Definitions of Determinant Factors:

- **Likelihood of delays**: Number of times that previous users reported delays over the total number of times that users used the offer [numeric value between 0 and 1; applicable to all modes of transport],
- **Traffic:** Measurement of real-time traffic (e.g. average speed, density of vehicles) [real value; applicable to trips by roads]
- **Last-minute changes:** Measurement of likelihood of last-minute changes from the TSP. This can be expressed as a probability from 0 to 1. These probability does not measure the delays deriving from the changes, which is measured by "Likelihood of delays" above [numeric value; bus, tram, rail, metro, air, urbanRail, others-drive-car],
- **Time of trip (rush vs non-rush hour):** Time of trip overlap to rush hours [real value],

Contract No. 881825

- **Frequency of service:** Number of scheduled trips per hour (depending on time and day of the trip). [numeric value; applicable to public transport modes]
- **Weather forecast (day of the trip):** Probability of rain at the day of the trip. To get the value the OpenWeather service is used [real value between 0 and 1; applicable to all modes].

4.2.3. CHEAP

Definitions of Determinant Factors:

- **Total price:** overall price of tickets in EUR (obtained from TRIAS file) covering connections included in the offer [real value, applicable to all modes],
- **Can share cost:** value 1 if for the ride-sharing offer is possible to share costs among several passengers and 0 otherwise [binary value; others-drive-car],
- **Ticket coverage:** a number between 0 and 1 representing a fraction of trip duration for which it is possible to book tickets [real value, all modes of transport].

4.2.4. COMFORTABLE

Definitions of Determinant Factors:

- **Number of interchanges:** Number of connections that constitute a given offer – 1 [integer value, applicable to offer],
- **User feedback:** average value of experienced comfort, collected as user feedback e.g. on five stars scale [numeric value; applicable to all modes of transport],
- **Cleanliness (station/stop/vehicle):** previously reported average value about experienced cleanliness (of stops, stations and vehicles) used by a given TSP, e.g. on five stars scale [numeric value; applicable to all public transport modes],
- **Seating quality:** previously reported average value about experienced seats comfort of vehicles used by a given TSP, e.g. on five stars scale [numeric value; applicable to all modes of transport],
- **Amount of space available, crowdedness:** previously reported average value about experienced available space in vehicles used by a given TSP, e.g. on five stars scale [numeric value; applicable to all modes of transport],
- **Presence of silence area:** value of 1 if there is a silence area in a given train, value of 0 otherwise [binary value, rail],
- **Privacy level:** previously reported average value about experienced privacy level in vehicles used by a given TSP, e.g. on five stars scale. [numeric value; all modes of transport],
- **Ticket coverage:** a number between 0 and 1 representing a fraction of trip duration for which it is possible to book tickets [real value, all modes of transport],
- **Ride smoothness:** smoothness (e.g. presence and intensity of sudden changes in the speed and direction of the vehicle) of the trip as perceived by the passenger. It is evaluated by the passenger on the five-star scale. [numeric value; all modes of transport],
- **Weather forecast (day of trip)**: Probability of rain at the day of the trip. To get the value the OpenWeather service is used [real value between 0 and 1; applicable to all modes].

Contract No. 881825

-

4.2.5. DOOR-TO-DOOR

Definitions of Determinant Factors:

- **Starting point:** latitude and-longitude coordinates of the starting point of the trip [double value (double array); applicable to all modes]
- **Ending point:** latitude and-longitude coordinates of the starting point of the trip [double value (double array); applicable to all modes]
- **Walking distance:** Manhattan distance between the starting/ending point of the trip and the public transport stop / ride-sharing meeting point [real value; applicable to all modes].

4.2.6. ENVIRONMENTALLY FRIENDLY

Definitions of Determinant Factors:

- **CO2 emission/km per traveller:** CO2 emissions per km and per traveller of this offer [numeric value; applicable to all modes],
- **Energy consumption:** of this offer (converted to Joules [*numeric value; applicable to all modes*],
- **NOx emission/km per traveler:** NOx emissions per km and per transport mode. It can be calculated from the energy consumption since it is almost linear with the fuel consumption [numeric value; applicable to all modes of transport],
- **Carbon footprint:** CO2 emissions of this offer divided by the number of travellers [real value; applicable to all modes],
- **Bike-on-board:** value of 1 if a bike can be loaded to the public transport vehicle, values of 0 – otherwise [binary value; bus, tram, rail, metro, air, urbanRail, others-drive-car].

4.2.7. SHORT

Definitions of Determinant Factors:

- **Distance covered:** length of all connections [real value; all modes of transport],
- **Number of stops:** number of stops made by all connections [integer value; all modes].

4.2.8. MULTITASKING

Definitions of Determinant Factors:

- **Amount of space available, crowdedness:** average value about experienced available space in vehicles used by a given TSP, collected as feedback from travellers e.g. on five stars scale [*numeric value; applicable to all modes of transport*],
- **Presence of silence area:** value of 1 if there is a silence area in a given train, value of 0 otherwise [binary value, rail],
- **Presence of business area:** value of 1 if there is a business area in a given train, value of 0 otherwise [binary value; rail, air],
- **Internet connection availability:** value of 1 if there is internet connection available and value of 0 otherwise [binary value; applicable to all modes of transport],

Contract No. 881825

- **Plugs/charging points:** value of 1 if plugs/charging points are available and value of 0 otherwise. [binary value; applicable to all public transport modes],
- **Privacy level:** previously reported average value about experienced privacy level in vehicles used by a given TSP, e.g. on five stars scale. [numeric value; all modes of transport].

4.2.9. PANORAMIC

Definitions of Determinant Factors:

- **Number of landscapes:** Number of landscapes that can be observed during the trip estimated by querying OpenStreetMap for the number of sightseeing points in a given radius around the path of the trip. [integer value; applicable only to road transport modes],
- **Number of historical sites:** Number of historical sites that can be observed during the trip, estimated by querying OpenStreetMap for the number of sightseeing points in a given radius around the path of the trip [integer value; applicable only to road transport],
- **Number of monuments:** Number of monuments that can be observed during the trip, estimated by querying OpenStreetMap for the number of sightseeing points in a given radius around the path of the trip [integer value; applicable only to road transport].

4.2.10. HEALTHY

Definitions of Determinant Factors:

- **Fraction of the trip by bike/walk:** Number of legs done by bike/walk over the total number of legs [real value between 0 and 1; applicable to all modes],
- **Length of fraction of trip by bike/walk**: Total sum of the norm distances between the starting and ending points done by bike/walk [real value; applicable to all modes].

4.3. Offers in IP4

This section provides an overview of the data format adopted in Shift2Rail IP4 to model the multi-modal travel offers computed for a user looking for multi-modal travel solutions. The analysis of this data format supported the definition of the Offer Categorizer architecture and the implementation of its components. In particular, the following objectives were pursued: (i) analyse the data format that the Offer Categorizer needs to process for its integration in the IP4 ecosystem, (ii) identify the information available in each multi-modal offer to compute the determinant factors presented in Section 4.2, and (iii) define an internal data model for the Offer Categorizer that could support the implementation of generic offer categorization procedures abstracting from the specific input data format.

The analysis of the data format was carried out in collaboration with the CFM partners during the 3rd and 4th Collaboration Meeting and through a set of dedicated meeting. First of all, an analysis of the currently used data format was performed. Then, two extensions to the data format were discussed and agreed to accommodate the specific use cases of Ride2Rail: on one hand, how to model Ridesharing offers was defined, on the other hand, the specific format of the input message sent by the *Shopping Orchestrator* to the *OfferEnricherAndRanker* was defined (cf. Section 3.3.2). In the following paragraph, the most relevant aspects of the data format are presented. More details on the IP4 terms and the underneath conceptualisation can be found in (RIDE2RAIL, Deliverable D2.4 - Final Conceptualization of Choice Criteria and Incentives, 2020).

4.3.1. Ride2RailScoringRequest

The IP4 data format to model multi-modal offers is an XML representation based on an extension of the TRIAS format. The considered set of extensions have been developed by CFM projects, mainly within COACTIVE[5], and are made available by Ride2Rail within the Offer Categorizer source code for reference[6].

In the IP4 ecosystem, a mobility request sent through the Travel Companion (personal application) by a user to the Shopping Orchestrator receives as a response a set of multi-modal offers. The Offer Categorizer aims at providing of the categorizations of the offers computed by the Shopping Orchestrator before returning them to the user. For this purpose, a Ride2RailScoringRequest[7] payload is sent to the Offer Categorizer. The Ride2RailScoringRequest contains a *TripRequest* structure describing the mobility request performed by the user and a *TripResponse* structure reporting a TRIAS *TripResponse* payload and modelling the computed multi-modal offers. In this section, we will describe the main concepts represented in the following snippet, a complete example is available on Github[8].

A *TripRequest* describes the mobility request specifying the requested departure and arrival locations, and, if provided by the user, the requested time of departure or arrival.

A TRIAS *TripResponse* is composed of a single *TripResponseContext* node providing general information about the locations (e.g., stop points), and multiple *TripResult* nodes describing the computed solutions. Each *TripResult* describes a possible multi-modal travel solution computed for the mobility request and can contain information about different offers provided by the TSPs to cover the same solution. A *TripResult* contains one *Trip* (i.e., the multi-modal travel solution), composed of *TripLeg*(s) (i.e., a portion of the solution

[5] https://projects.shift2rail.org/s2r_ip4_n.aspx?p=CO-ACTIVE
[6] https://github.com/Ride2Rail/trias-extractor/tree/main/extensions
[7] https://github.com/Ride2Rail/trias-extractor/blob/main/extensions/ride2rail
[8] https://github.com/Ride2Rail/trias-extractor/blob/main/tests/test_example.xml

associated to a single transportation mode), and a set of *Tickets* associated to the *TripLeg*(s) composing the *Trip* (i.e., the offers that can be purchased to perform the solution).
A *TripLeg* can be of two different types:
TimedLeg (timetabled trip leg, e.g., train/bus/…)
ContinuousLeg (trip leg as movement with a continuously available mode, e.g., walk/car/…)
An item of the offer is represented by a *Ticket* that uses a *coactive:OfferItemTicketExtension* and can be used to cover one or more *TripLegs*(s). A complete offer for the entire *Trip* is represented by a *META-Ticket* that uses a *coactive:MetaTicketExtension* to represent the aggregated details of the offer and refers to multiple offer items (*Tickets*) covering the various *TripLegs*.
As a result, the entities that need to be categorized by the Ride2Rail Offer Categorizer are the *META-Tickets* (Offers) contained in each *TripResult*. The *coactive:OfferId* nodes contain the identifiers of the offers to be categorized.

As commented above, more than one *META-Ticket* (Offer) can be defined within a *TripResult* (aggregating different *Tickets* for the same *Trip*, e.g., one considering first class tickets and the other one economy tickets for the same *Trip*). Therefore, multiple offers can be associated with the same *Trip*. The computation performed by the Offer Categorizer considers both the offer features that can be computed accessing the *Trip* information and the ones directly contained in the *META-Ticket.*

Multi-modal offers including ridesharing can be defined using the already available structures in TRIAS. *TripLegs* representing a ride-sharing portion of the computed travel solution are modelled as a *ContinuousLeg* with transportation mode of type *others-drive-car* and further details contained in the *SharingService* node (data about the driver and the vehicle). Ridesharing offer items can be modelled as done for other *Tickets* referencing the ridesharing *TripLeg*.

Additional details on the data extracted from the Ride2RailScoringRequest are presented in the *trias-extractor* component description in Section 5.4.1. The specification of the internal data model adopted by the offer categorizer to compute the categorization is discussed in Section 5.2.

4.3.2. Additional Offer Features

The set of offer features that can be extracted from the data format adopted in IP4 for multi-modal offers is limited. For this reason, Ride2Rail defined in collaboration with the CFM a mechanism to enable the enrichment of the offers with additional information. These additional offer features are not mandatorily required for a TSP joining the IP4 ecosystem but, if provided, guarantee to the TSP a better categorization of its offers when proposed to the users.
Two mechanisms can be used by a TSP to provide these additional offer features: the *coactive:OfferItemContext* and the TRIAS *Attribute* nodes.

A *coactive:OfferItemTicketExtension,* used to define offer items extending the TRIAS *Ticket*, can have one or more *coactive:OfferItemContext* child nodes. Each node has *Code* and *Value* child nodes representing generic key-value pairs that can be used to describe the *OfferItem* (e.g., quality of travel, coach/seat code, train code, etc.). This solution is recommended if the additional offer feature provided by the TSP is directly associated with the offer item and/or with all the *TripLeg*s covered by the offer item. Figure 3 shows how additional features and their values are encoded into the *OfferItemContext*.

```
<coactive:OfferItemContext
  <coactive:Code>passenger_feedback</coactive:Code>
  <coactive:Value>4.2</coactive:Value>
</coactive:OfferItemContext>
```

Figure 3 An additional feature (passenger_feedback) and its value encoded into the OfferItemContext.

The second mechanism is recommended for additional offer features associated with a specific *TripLeg*. In this case, the *Attribute* nodes in the *TripLeg Service* description can be used providing a key-value mechanism similar to the one of the *coactive:OfferItemContext* as shown in Figure 4.

```
<ns3:Attribute>
  <ns3:Code>likelihood_of_delays</ns3:Code>
  <ns3:Text>
    <ns3:Text>0.66</ns3:Text>
  </ns3:Text>
</ns3:Attribute>
```

Figure 4 A feature encoded as an Attribute to a TripLeg.

The *trias-extractor* component, discussed in Section 5.4.1, implements also an additional mechanism to parse the additional offer features. This third mechanism is based on composite keys (*key:leg_id*) and allows to specify through the *coactive:OfferItemContext* also information for a specific *TripLeg*. This option can be used by the TSPs to provide all the additional offer features through the *coactive:OfferItemContext*. An example is shown in Figure 5.

Contract No. 881825

```
<coactive:OfferItemContext>
  <coactive:Code>likelihood_of_delays:ID344352-4432-44</coactive:Code>
  <coactive:Value>0.66</coactive:Value>
</coactive:OfferItemContext>
```

Figure 5 A feature encoded into the OfferItemContext as a key-value pair.

The description of the additional information that can be provided by a TSP through the *coactive:OfferItemContext* and/or the *Attribute*(s) can be found in Section 4.4.1.

4.4. Data sources

This section describes the data sources considered to develop and test the Offer Categorizer.

4.4.1. TRIAS Examples

The integration of the Offer Categorizer into the IP4 ecosystem has as a requirement the ability to process the IP4 data format described in Section 5.3. For this reason, in collaboration with the CFM partners, we identified a dataset of potential *Ride2RailScoringRequest* to test the categorization process. The examples were generated considering various mobility requests in the Lisbon area (demonstration site of the Shift2MaaS project[9]).

At the time of implementing the Offer Categorizer, the data structure defined by Ride2Rail, such as the additional offer features to be provided by TSPs, and the data about the ridesharing offers, were not available in the IP4 ecosystem. To enable the testing of all Offer Categorizer functionalities, we implemented the software module Trias-example-generator enriching the TRIAS files by synthetic data. The source code of the module is available on GitHub[10]. The names of determinant factors that are assigned by the Trias-example-generator module to the TRIAS file together with the numeric type of the values, lower and upper bound and applicable modes of transport are presented in Table 4.

[9] https://www.shift2maas.eu/

[10] https://github.com/Ride2Rail/trias-example-generator/wiki

Contract No. 881825

It is assumed that in practice some values of determinant factors might not be available due to the lack of data. Determinant factors are generated independently. By default, the probability to generate a determinant factor is set to the value of 0.25 for all determinant factors. This value can be changed separately for each determinant factor by modifying the script TSP_data_generator.py.

Table 4 Overview of determinant factors imputed by the Trias-example-generator module to the TRIAS file.

Name of the determinant factor	Numeric type, lower bound, upper bound	Applicable modes of transport
likelihood_of_delays	real value, 0, 1	all
last_minute_changes	real value, 0, 1	bus, tram, rail, metro, air, urbanRail, others-drive-car (ridesharing)
frequency_of_service	Integer value, 1, 20	bus, tram, rail, metro, air, urbanRail
user_feedback	real value, 1, 5	all
cleanliness	real value, 1, 5	bus, tram, rail, metro, air, urbanRail, others-drive-car (ridesharing)
seating_quality	real value, 1, 5	all
space_available	real value, 1, 5	all
silence_area_presence	integer value, 0, 1	rail
privacy_level	real value, 1, 5	all
bike_on_board	integer value, 0, 1	bus, tram, rail, metro, air, urbanRail, others-drive-car(ridesharing)
business_area_presence	integer value, 0, 1	rail, air
internet_availability	integer value, 0, 1	bus, tram, rail, metro, air, urbanRail, others-drive-car (ridesharing)
plugs_or_charging_points	integer value, 0, 1	bus, tram, rail, metro, air, urbanRail, others-drive-car (ridesharing)
number_of_persons_sharing_trip	integer value, 0, 20	others-drive-car (ridesharing)
shared_with_other_passengers	integer value, 0, 1	others-drive-car (ridesharing)
safety_features	real value, 1, 5	bus, tram, rail, metro, air, urbanRail, others-drive-car (ridesharing)
vehicle_age	integer value, 0, 25	others-drive-car (ridesharing)
passenger_feedback	real value, 1, 5	bus, tram, rail, metro, air, urbanRail, others-drive-car (ridesharing)

Contract No. 881825

ride_smoothness	real value 1, 5	bus, tram, rail, metro, air, urbanRail, others-drive-car (ridesharing)
certified_driver	integer value, 0, 1	others-drive-car (ridesharing)
driver_license_issue_date	date, 1960-1-1T00:00:00.000Z, 2020-1-1T00:00:00.000Z	others-drive-car (ridesharing)
repeated_trip	integer value, 0, 1	others-drive-car (ridesharing)

The values of determinant factors are added to the corresponding Ticket under Extension element as a new *coactive:OfferItemContext* element.
To establish a set of test cases from the TRIAS examples provided by the CFM partners, we prepared three subsets of examples. Each example corresponds to one Ride2RailScoringRequest. The selection of examples followed the requirement to cover the diversity of modes of transport and to cover various numbers of available offers. The parameters of selected TRIAS files are presented in Table 5.

Table 5 Parameters of test cases selected from TRIAS examples provided by the CFM projects to the Ride2Rail.

Name of the test case file	Number of trip results	Number of all tickets	Available mode of transport (number of occurrences in all trip legs)									
			Rail	Metro	Bus	Unknown	Urban rail	Air	Tram	Walk	Cycle	Others_drive_car
Subset 1 - selected from TRIAS files received from CFM partners on 18.05.2021												
subset_1_no_3.xml	3	11	0	0	3	0	1	0	0	3	1	0
subset_1_no_9.xml	2	4	0	0	2	1	0	0	0	1	1	0
subset_1_no_12.xml	3	15	0	0	4	3	0	0	0	2	2	0
subset_1_no_18.xml	5	9	0	0	4	1	0	0	0	7	4	0
subset_1_no_21.xml	5	59	0	0	12	0	0	0	0	8	0	0
subset_1_no_22.xml	5	35	0	0	10	0	0	0	0	4	1	0
subset_1_no_26.xml	4	7	0	0	3	0	0	0	0	4	2	0
subset_1_no_37.xml	0	0	0	0	0	0	0	0	0	0	0	0
subset_1_no_44.xml	5	9	0	0	4	2	0	0	0	4	4	0
subset_1_no_50.xml	5	35	0	0	8	0	0	0	2	4	3	0
Subset 2 - selected from TRIAS files received from CFM partners on 31.05.2021												
subset_2_no_1.xml	3	6	0	0	0	0	3	0	0	3	0	0
subset_2_no_2.xml	2	3	0	0	0	1	0	0	0	3	1	0
subset_2_no_3.xml	3	15	0	0	4	0	0	0	0	8	2	0
subset_2_no_4.xml	5	8	0	0	4	0	0	0	0	5	0	0

Contract No. 881825

subset_2_no_5.xml	5	8	0	0	4	0	0	0	0	9	0	0
subset_2_no_6.xml	3	3	0	0	0	3	0	0	0	3	0	0
subset_2_no_7.xml	5	8	0	0	4	0	0	0	0	5	0	0
subset_2_no_8.xml	1	1	0	0	0	0	0	0	0	1	0	0
subset_2_no_9.xml	2	12	0	0	2	2	0	0	0	6	2	0
subset_2_no_10.xml	2	4	0	0	1	2	1	0	0	0	0	0
Subset 3 - test cases containing ridesharing leg (derived from Subset2 files Nr. 3, 5 and 9)												
subset_3_no_3.xml	3	15	0	0	3	0	0	0	0	8	2	1
subset_3_no_5.xml	5	8	0	0	3	0	0	0	0	9	0	1
subset_3_no_9.xml	2	12	0	0	1	2	0	0	0	6	2	1

Subset 1 was compiled from TRIAS files obtained from CFM partners. As the examples in Subset 1 do not contain complete information about the request, the second set of examples was demanded from CFM partners and received on May 31, 2021. From the second set, we selected the other ten examples that form Subset 2. At this stage of the Ride2Rail project, there is no available real-world ridesharing data. To be able to test the computation of determinant factors that are specific to ridesharing, we selected from Subset 2 three examples, Nr. 3, 5 and 9, and we replaced manually one bus leg with a ridesharing leg. The resulting test cases constitute Subset 3.

As the resulting XML files may contain potentially sensitive information about the file formats or information that is owned by TSPs, the resulting files have been uploaded to a private GitHub repository trias-generator-xml-examples[11], where they are currently available to Ride2Rail partners.[12]

Similarly, as there is no real-world TSP data that could be used to define the values of TSP related determinant factors, to be able to test the computations performed by the Offer Categorizer, we created from each test case three instances by imputing randomly generated TSP data. Instances differ by the density of TSP data. We selected three probability values, 0.25, 0.5 and 0.75, respectively. Independently for each determinant factor listed in Table 4, we generated with this probability randomly a value of the determinant factor while considering its numeric type and lower and upper bounds. Thus, for each test case, we obtained three test cases enriched by synthetic TSP data. Files containing randomly generated TSP data are available in directories: subset_1_enriched, subset_2_enriched and subset_3_enriched.

[11] https://github.com/Ride2Rail/trias-generator-xml-examples

[12] Access to the repository can be obtained upon request from Cristian Consonni (email: cristian.consonni@eurecat.org). The files are organized in folders, subset_1, subset_2 and subset_3.

4.4.2. MoTiV data

Another dataset we have used to develop our system was the MoTiV[13] dataset, which is publicly available on Zenodo.[14] The MoTiV (Mobility and Time Value) dataset contains data about travellers and their journeys, collected from a dedicated mobile application developed within the scope of the project, called Woorti. MoTiV data have been collected by users in 13 countries over a period of 8 months, from February 2019 to December 2019. Each trip contains multi-faceted information: from the transport mode, through its evaluation, to the positive/negative experience factors.

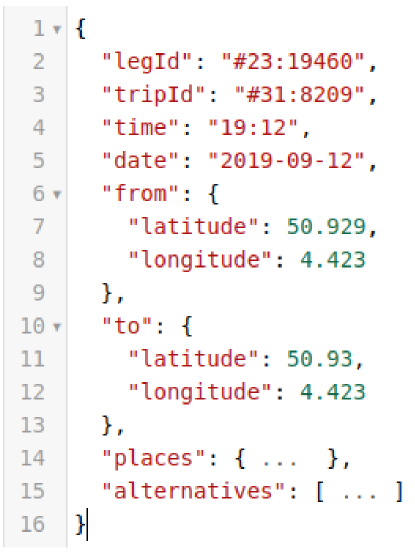
As part of the MoTiV project, data were collected from a journey planning engine called RouteRANK[15]. Figure 6 below shows the trip data that were sent to the RouteRANK engine: leg and trip identifier (`legId, tripId`), trip date (`date`) and time (`time`) and starting (`from`), and ending point coordinates (`to`).

[13] MoTiV (Mobility and Time Value) was an EU Horizon 2020 project which addressed the emerging perspectives on changing Value of Travel Time (VTT). MoTiV explored the dynamics of individual preferences, behaviors and lifestyles that influence travel and mobility choices. In other words, the aim of the project was to answer the question "what does value of travel time mean for the end user's travel experience?"

The MoTiV project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 770145.

[14] Cristian Consonni, Silvia Basile, Matteo Manca, Ludovico Boratto, Ghadir Pourhashem, Tatiana Kovacikova, André Freitas, & Yannick Cornet. (2020). MoTiV: a Dataset of European User Mobility for Behavioral-Data (v1.0) [Data set]. 15th International AAAI Conference on Web and Social Media (ICWSM 2021). Zenodo. https://doi.org/10.5281/zenodo.4027465

[15] RouteRANK website https://www.routerank.com/ [accessed on 2021-09-22].

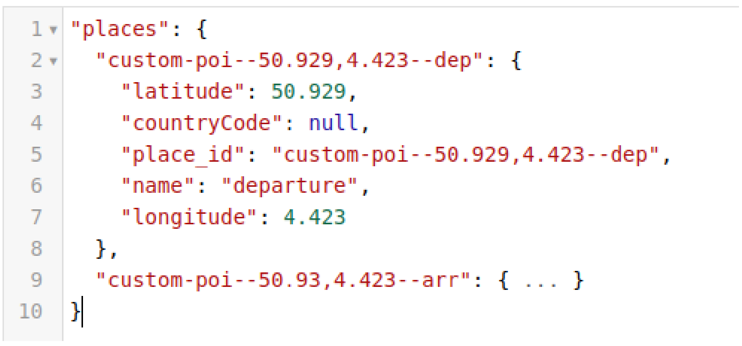


```
{
  "legId": "#23:19460",
  "tripId": "#31:8209",
  "time": "19:12",
  "date": "2019-09-12",
  "from": {
    "latitude": 50.929,
    "longitude": 4.423
  },
  "to": {
    "latitude": 50.93,
    "longitude": 4.423
  },
  "places": { ...  },
  "alternatives": [ ... ]
}
```

Figure 6 Data sent to the RouteRANK engine.

The engine responds with:

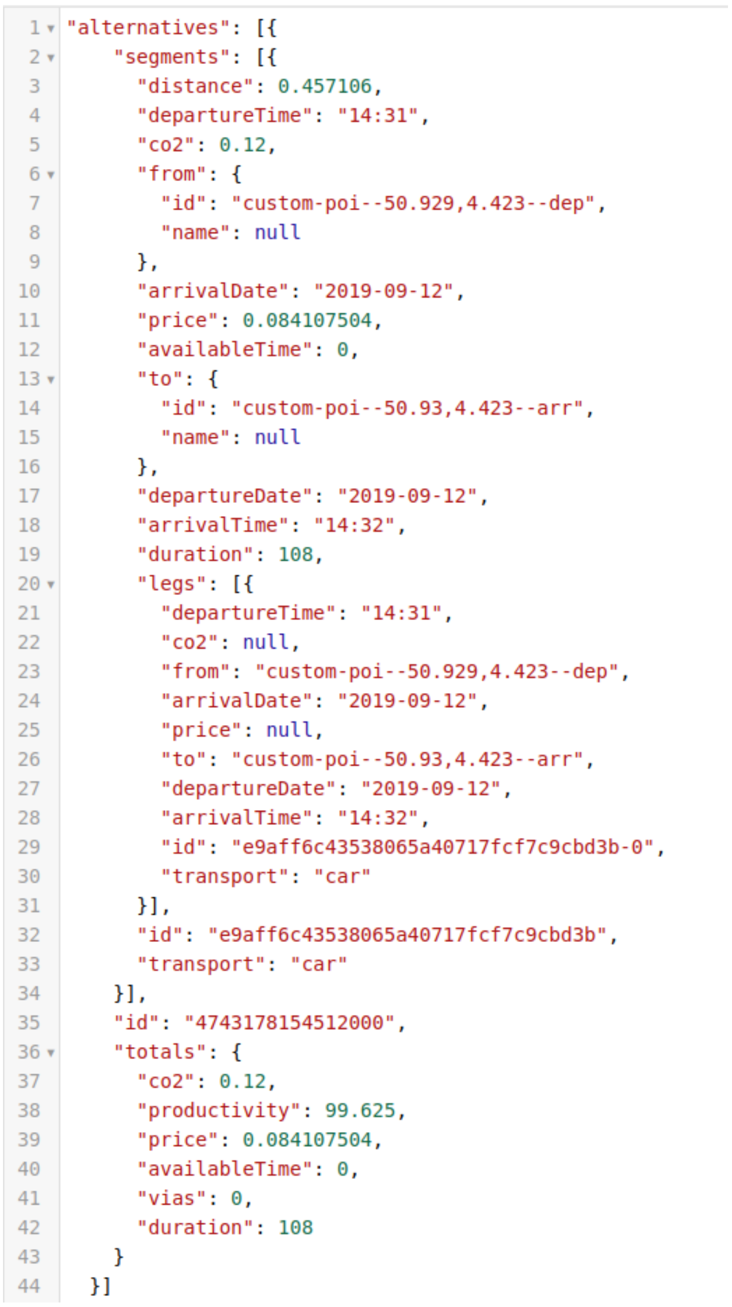
- a list of places, that can be named street addresses of custom points-of-interest (POIs) as shown by Figure 7.



```
"places": {
  "custom-poi--50.929,4.423--dep": {
    "latitude": 50.929,
    "countryCode": null,
    "place_id": "custom-poi--50.929,4.423--dep",
    "name": "departure",
    "longitude": 4.423
  },
  "custom-poi--50.93,4.423--arr": { ... }
}
```

Figure 7 Data (list of places) sent to the RouteRANK engine.

- a list of alternatives, that are trips that the users can take, these trips are organized in segments and legs. Each alternative is presented with additional information such as CO_2 emissions and price. An example is shown in Figure 8.



```
"alternatives": [{
    "segments": [{
      "distance": 0.457106,
      "departureTime": "14:31",
      "co2": 0.12,
      "from": {
        "id": "custom-poi--50.929,4.423--dep",
        "name": null
      },
      "arrivalDate": "2019-09-12",
      "price": 0.084107504,
      "availableTime": 0,
      "to": {
        "id": "custom-poi--50.93,4.423--arr",
        "name": null
      },
      "departureDate": "2019-09-12",
      "arrivalTime": "14:32",
      "duration": 108,
      "legs": [{
        "departureTime": "14:31",
        "co2": null,
        "from": "custom-poi--50.929,4.423--dep",
        "arrivalDate": "2019-09-12",
        "price": null,
        "to": "custom-poi--50.93,4.423--arr",
        "departureDate": "2019-09-12",
        "arrivalTime": "14:32",
        "id": "e9aff6c43538065a40717fcf7c9cbd3b-0",
        "transport": "car"
      }],
      "id": "e9aff6c43538065a40717fcf7c9cbd3b",
      "transport": "car"
    }],
    "id": "4743178154512000",
    "totals": {
      "co2": 0.12,
      "productivity": 99.625,
      "price": 0.084107504,
      "availableTime": 0,
      "vias": 0,
      "duration": 108
    }
  }]
```

Figure 8 Alternatives from the RouteRANK engine.

RouteRANK trip alternatives data are mapped to the corresponding TRIAS field and parsed so that they can be used to test the OC as if they were data coming from the S2 ecosystem,[16] this mapping is detailed in Appendix A.

4.5. Quantification of offer categories

4.5.1. Assumptions and prerequisites

In this section, we describe the main assumptions and conditions under which the OC is operational and which outputs are provided in exceptional situations. The OC applies the following rules:

- if at least two trip offers are returned in response to the mobility request, the categorization of the offer is performed,
- if at least one determinant factor is available for an offer category, it is sufficient to proceed with quantification of the offer category,
- feature collectors compute the values of determinant factors in a way that higher value corresponds to a higher positive contribution of the determinant factor to relevant offer categories,
- offer categories are quantified in a way that higher value assigned to an offer category corresponds to the more positive evaluation by a typical passenger.

For the full operation of the Offer Categorizer the following prerequisites need to be satisfied:

- Properly formatted multi-modal end-to-end offers need to be provided as input. For the integration in IP4, the implemented Offer Categorizer expects as input a Ride2RailScoringRequest message containing information about the mobility request (including parameters of the mobility request) and multi-modal offers computed as a response to the mobility request (see section 4.3). To obtain a
- For more accurate offer categorization, the multi-modal offers should be enriched by TSP with additional data about the provided service (currently this data is mocked, see section 4.4.1).
- Feature collectors need to be able to access external services (APIs and web services) they rely on.
- The following service need to be hosted: OpenStreetMap server supplied with data that cover the area coinciding with the coverage of transport services.

[16] Access to the routerRANK can be obtained upon request from Cristian Consonni (email: cristian.consonni@eurecat.org).

Contract No. 881825

4.5.2. Methodology

Here, we assume as an input TRIAS file containing a response from the IP4 ecosystem to a mobility request. The TRIAS file has already been enriched by the TSP data (currently in the development and for the testing, we used the synthetic TSP data (see Section 4.4.1), nevertheless, when the Offer categorizer will be deployed in the IP4 ecosystems, the data will be enriched by Offer Enricher.

Furthermore, during the development of the OC we used the MoTiV data, described in Section 4.4.2, as input. These data were parsed and mapped to the same structure as the TRIAS data and therefore they are subsequently processed in the same way.

Considering these inputs, quantification of offer categories is executed in several phases:

Phase 1: If a responsible feature collector requires some external services (e.g., exchange rates to convert prices to EUR, weather forecast to estimate trip comfort, etc.) an external service is used to obtain such data.

Phase 2: If a feature collector processes a determinant factor having multiple values (there might be assigned one value for each trip leg belonging to a trip offer), values assigned to trip legs are aggregated based on a factor specific formula.

Phase 3: Further, the feature collector applies a standardization procedure to combine values of multiple determinant factors belonging to one offer category.

Phase 4: Finally, standardized values of determinant factors allocated to one offer category are aggregated by using surrogate weights derived from the rank-order of determinant factors within each offer category.

Phases 1-3 are performed in parallel by individual feature collectors and the results are stored to the offer cache. The computation is concluded by Offer Categorizer Core that is executing Phase 4.

Phase 1: Use of external services

For more details about the external services implemented by individual feature collectors, see Section 5.5.

Phase 2: Aggregation of determinant factors over trip legs

In the simplest cases, to aggregate the values over trip legs, the summation of values can be applied. This applies to additive quantities such as price, duration of the trip, carbon footprint-related determinant factors, etc. For some cases, the summation is not applicable. For example, if the values of determinant factors are bounded, e.g., values quantified on the five-star scale, we cannot sum them up, but averaging of values needs to be applied. Here, we apply weighted average. To define mathematically how the aggregated values are calculated, we introduce the notation:

M - Number of trip legs with known and properly defined values of a determinant factor belonging to an offer,

w_i - value of a weight determining the importance of trip leg $i = 1, \dots, M$ (typically, in calculations, the duration of a trip leg is used as a weight),

q_i - value of a determinant factor associated with trip leg $i = 1, \dots, M$.

The aggregated value, v, of the determinant factor is calculated as $v = \frac{\sum_{i=1}^{M} w_i q_i}{\sum_{i=1}^{M} w_i}$.

Contract No. 881825

Phase 3: Standardization of determinant factors

The range of values of determinant factors can be significantly different. This problem is closely related to the data normalization and standardization in statistics and data analysis. The aim of standardization is to facilitate the comparison between determinant factors by changing the range of values. To perform the standardization of determinant factors, two approaches have been implemented:

- z-score,
- min-max score.

To define mathematically how the values of the z-score and min-max score are calculated, we introduce the notation:

N - number of offers with known values of a determinant factor belonging to a mobility request,

v_i - value of a determinant factor associated with offer $i = 1, \dots, N$,

μ - mean value of the v_i values for $i = 1, \dots, N$,

σ - standard deviation of the v_i values for $i = 1, \dots, N$,

$v^{min} = {}_{i=1,\dots,N}^{\ \ min}\{v_i\}$ - minimum value of a determinant factor,

$v^{max} = {}_{i=1,\dots,N}^{\ \ max}\{v_i\}$ - maximum value of a determinant factor.

The value of the z-score, $s_i^{z-score}$, for a determinant factor associated with the offer $i = 1, \dots, N$ is calculated following the formula $s_i^{z-score} = \frac{v_i - \mu}{\sigma}$. Analogously, the value of the z-score, s_i^{minmax}, for a determinant factor associated with the offer $i = 1, \dots, N$ is calculated following the formula $s_i^{minmax} = \frac{v_i - v^{min}}{v^{max} - v^{min}}$.

Both calculations assume that the higher is the value of the score associated with determinant factor, the better it is for a passenger. In an opposite case, the values of the score should be subtracted from the value of 1, i.e. $s_i^{z-score} = 1 - \frac{v_i - \mu}{\sigma}$ and $s_i^{z-score} = 1 - \frac{v_i - v^{min}}{v^{max} - v^{min}}$. This way score assigned to determinant factors are aligned in terms of the range, but also in terms of the interpretation of values.

Which type of the score (z-score or min-max) should be applied can be specified in the configuration file of the feature collector.

Phase 4: Computation of offer categories

The purpose of this phase is to compute the numeric value quantifying an offer category based on the values of determinant factors calculated in Phase 3. The allocation of determinant factors to offer categories is described in Section 5.1. The same methodology has been applied to all offer categories, and for this reason, we explain the computation for one offer category. Let us assume the following notation:

D - number of determinant factors with known values computed in phase 3 that are associated with an offer category,

s_i – the value of the determinant factor $i = 1, \dots, D$, calculated in phase 3.

r_i - rank assigned to the determinant factor $i = 1, \dots, D$, while considering its importance within the set of determinant factors associated with the offer category,

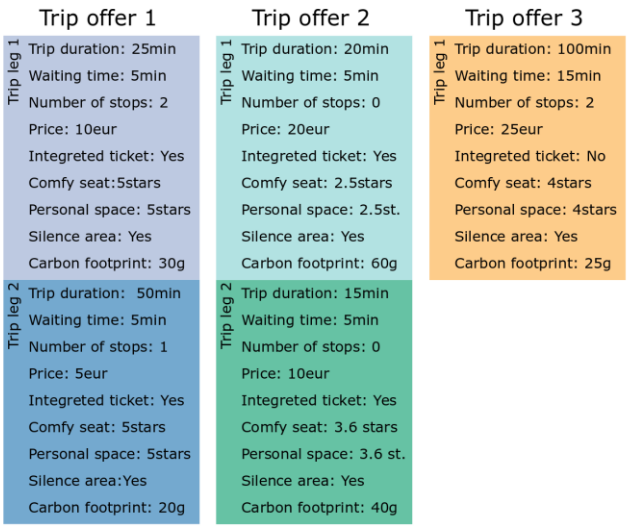
w_i - weight determining the importance of the determinant factor $i = 1, \dots, D$ within the set of determinant factors associated with the offer category.

The value of the offer category, C, is calculated as $C = \sum_{i=1}^{D} s_i w_i$.

The values of weights w_i are determined based on the rank values r_i considering the Rank order distribution (ROD) weights (Roberts & Goodwin, 2002).

4.5.3. Example

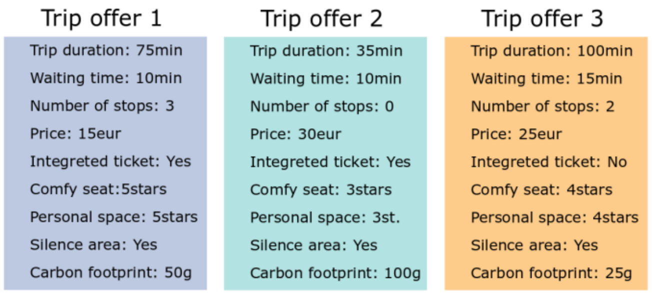
To demonstrate the calculation of offer categories, we consider the following example with three trip offers, five trip legs, and values of determinant factors visualized in Figure 9.



	Trip offer 1	Trip offer 2	Trip offer 3
Trip leg 1	Trip duration: 25min	Trip duration: 20min	Trip duration: 100min
	Waiting time: 5min	Waiting time: 5min	Waiting time: 15min
	Number of stops: 2	Number of stops: 0	Number of stops: 2
	Price: 10eur	Price: 20eur	Price: 25eur
	Integreted ticket: Yes	Integreted ticket: Yes	Integreted ticket: No
	Comfy seat:5stars	Comfy seat: 2.5stars	Comfy seat: 4stars
	Personal space: 5stars	Personal space: 2.5st.	Personal space: 4stars
	Silence area: Yes	Silence area: Yes	Silence area: Yes
	Carbon footprint: 30g	Carbon footprint: 60g	Carbon footprint: 25g
Trip leg 2	Trip duration: 50min	Trip duration: 15min	
	Waiting time: 5min	Waiting time: 5min	
	Number of stops: 1	Number of stops: 0	
	Price: 5eur	Price: 10eur	
	Integreted ticket: Yes	Integreted ticket: Yes	
	Comfy seat: 5stars	Comfy seat: 3.6 stars	
	Personal space: 5stars	Personal space: 3.6 st.	
	Silence area:Yes	Silence area: Yes	
	Carbon footprint: 20g	Carbon footprint: 40g	

Figure 9 Values of determinant factors of trip offers used to illustrate quantifications performed by the offer categorizer.

First, the values of determinant factors are aggregated over trip legs belonging to the same trip offer. In the majority of the cases (Trip duration, Waiting time, Number of stops, Price, Carbon footprint), determinant factors are additive. Hence, the aggregated value is calculated as the sum of individual values. Otherwise, we apply the weighted average, while using factor Trip durations as a weight. We can represent binary values as 0 and 1 as 0 and 1 and apply weighted average as well. After aggregating the values over trip legs, we obtain the values presented in Figure 10.



Trip offer 1	Trip offer 2	Trip offer 3
Trip duration: 75min	Trip duration: 35min	Trip duration: 100min
Waiting time: 10min	Waiting time: 10min	Waiting time: 15min
Number of stops: 3	Number of stops: 0	Number of stops: 2
Price: 15eur	Price: 30eur	Price: 25eur
Integreted ticket: Yes	Integreted ticket: Yes	Integreted ticket: No
Comfy seat:5stars	Comfy seat: 3stars	Comfy seat: 4stars
Personal space: 5stars	Personal space: 3st.	Personal space: 4stars
Silence area: Yes	Silence area: Yes	Silence area: Yes
Carbon footprint: 50g	Carbon footprint: 100g	Carbon footprint: 25g

Figure 10 Values of determinant factors of trip offers obtained by aggregating the values of determinant factors of trip legs presented in Figure 9.

Contract No. 881825

The next step is the standardization of values of determinant factors, where the min-max or z-score approach can be applied. When applying min-max approach, the standardized value of the determinant factor Trip duration, *TD*, for Trip offer 1 is calculated as follows:

$$TD_{offer=1}^{\min\max-standardized} = \frac{TD_{offer=1} - \min\{TD_{offer=1}, TD_{offer=2}, TD_{offer=3}\}}{\max\{TD_{offer=1}, TD_{offer=2}, TD_{offer=3}\} - \min\{TD_{offer=1}, TD_{offer=2}, TD_{offer=3}\}}$$

$$= \frac{75 - \min\{75,35,100\}}{\max\{75,35,100\} - \min\{75,35,100\}} = \frac{75-35}{100-35} = 0.615.$$

Calculation of z-score requires the average and standard deviation

$$TD^{avg} = \frac{TD_{offer=1} + TD_{offer=2} + TD_{offer=3}}{3} = \frac{75+35+100}{3} = 70,$$

$$TD^{std} = \sqrt{\frac{\left(TD_{offer=1} - TD^{avg}\right)^2 + \left(TD_{offer=2} - TD^{avg}\right)^2 + \left(TD_{offer=3} - TD^{avg}\right)^2}{2}}$$

$$= \sqrt{\frac{(75-70)^2 + (35-70)^2 + (100-70)^2}{2}} = 32.787.$$

Then, when applying z-score approach, the standardized value of the determinant factor Trip duration, for Trip offer 1 is calculated as

$$TD_{offer=1}^{z-score-standardized} = \frac{TD_{offer=1} - TD^{avg}}{TD^{std}} = \frac{75-70}{32.787} = 0.153.$$

Applying the same methodology to values of all determinants factor presented in Figure 9, we obtain the standardized values presented in Table 6.

Table 6 Standardized values of determinant factors presented in Figure 9. Both scores, min-max and z-score, are presented.

Determinant factor	min-max			z-score		
	Trip offer 1	Trip offer 2	Trip offer 3	Trip offer 1	Trip offer 2	Trip offer 3
Trip duration	0.615	0.000	1.000	0.153	-1.068	0.915
Waiting time	0.000	0.000	1.000	-0.577	-0.577	1.155
Number of stops	1.000	0.000	0.667	0.873	-1.091	0.218
Price	0.000	1.000	0.667	-1.091	0.872	0.218
Integrated ticket	1.000	1.000	0.000	0.577	0.577	-1.155
Comfy seat	1.000	0.000	0.500	1.000	-1.000	0.000
Personal space	1.000	0.000	0.500	1.000	-1.000	0.000
Silence area	1.000	1.000	1.000	1.000	1.000	1.000
Carbon footprint	0.333	1.000	0.000	0.333	1.000	0.000

To illustrate the computation of offer categories, from the standardized values, we need to introduce considered offer categories and the association of determinant factors with offer categories. For illustration purposes, we consider four offer categories: Quick, Cheap, Comfortable and Environmentally friendly. Determinant factors are, in this example, associated with offer categories in the following way:

- Quick (Trip duration, Waiting time, Number of stops),
- Cheap (Price, Integrated tickets),
- Comfortable (Comfy seat, Personal space, Silence area),
- Environmentally friendly (Carbon footprint).

Next, ranks indicating the importance of determinant factors for a given offer category, need to be assigned. In the illustration we consider ranks presented in Table 7.

Table 7 Association of determinant factors with offer categories and ranking of determinant factors within each offer category.

Offer categories	Determinant factor	Ranking
Quick	Trip duration	1
	Waiting time	3
	Number of stops	2
Cheap	Price	1
	Integrated ticket	2
Comfortable	Comfy seat	1
	Personal space	2
	Silence area	3
Env. friendly	Carbon footprint	1

As a next step, the way how the standardized values of determinant factors presented in Table 7 contribute to the offer category needs to be analysed. The standardized values of determinant factors should be aligned in a way that the larger the value, the more it contributes to the offer category. If the relationship is maintained, the values of the determinant factor remain the same. Otherwise, it is flipped by subtracting its values from the value of 1. For example, the higher is the values of Comfy seat, the more it contributes to the Comfortable category. Hence the value is maintained. In the opposite case, the higher the trip duration, the less it contributed to the Quick category. Thus, the value of trip duration gets flipped. After this step, the values of standardized determinant factors presented in Table 7, are transformed to values in
Table 8.

Table 8 Transformed standardized values of determinant factors, after considering their relationship with offer categories.

Determinant factor	min-max			z-score		
	Trip offer 1	Trip offer 2	Trip offer 3	Trip offer 1	Trip offer 2	Trip offer 3
Trip duration	0385	1.000	0.000	0.848	2.068	0.085
Waiting time	1.000	1.000	0.000	1.577	1.577	-0.155
Number of stops	0.000	1.000	0.333	0.127	2.091	0.782
Price	1.000	0.000	0.333	2.091	0.127	0.782
Integrated ticket	1.000	1.000	0.000	0.577	0.577	-1.155

Comfy seat	1.000	0.000	0.500	1.000	-1.000	0.000
Personal space	1.000	0.000	0.500	1.000	-1.000	0.000
Silence area	1.000	1.000	1.000	1.000	1.000	1.000
Carbon footprint	0.667	0.000	1.000	0.667	0.000	1.000

For computing the values of offer categories from the values of determinant factors, the rank order distribution (ROD) weights are available. In Table 9 we provide the ROD weights (see description of Phase IV) for 2-10 determinant factors available for an offer category. Values of weights for higher number of determinant factors need to be derived from formulas the ROC weights.

Table 9 Values of rank order distribution (ROD) weights for 2-10 determinant factors.

Rank	Number of determinant factors available for an offer category								
	2	3	4	5	6	7	8	9	10
1	0.6932	0.5232	0.4180	0.3471	0.2966	0.2590	0.2292	0.2058	0.1867
2	0.3068	0.3240	0.2986	0.2686	0.2410	0.2174	0.1977	0.1808	0.1667
3		0.1528	0.1912	0.1955	0.1884	0.1781	0.1672	0.1565	0.1466
4			0.0922	0.1269	0.1387	0.1406	0.1375	0.1332	0.1271
5				0.0619	0.0908	0.1038	0.1084	0.1095	0.1081
6					0.0445	0.0679	0.0805	0.0867	0.0893
7						0.0334	0.0531	0.0644	0.0709
8							0.0263	0.0425	0.0527
9								0.0211	0.0349
10									0.0173

Considering the min-max scores of determinant factors presented in Table 8, the ranking of determinant factors presented in Table 7 and the ROD weights presented in Table 9 we can calculate the offer category Quick for trip offer 1 as:
$Quick_{offer=1}^{ROD} = \sum_{i=1}^{3} w_i s_i = 0.5232 * 0.385 \ + 0.3240 * 0.000 \ + 0.1528 * 1.000 \ = 0.354.$

Considering the z-scores of determinant factors presented in Table 8, the ranking of determinant factors presented in Table 7 and the ROD weights presented in Table 9 we can calculate the offer category Quick for trip offer 1 as:
$Quick_{offer=1}^{ROD} = \sum_{i=1}^{3} w_i s_i = 0.5232 * 0.845 \ + 0.3240 * 0.127 \ + 0.1528 * 1.578 \ = 0.725.$

Values obtained for other offer categories are presented in Table 10. The top values of scores are highlighted. From the pattern created by the highlighted cells, we see that both scoring schemas and both weighting schemas result in a similar order of trip offers. By inspecting the parameters of Trip offers presented in Figure 9, we can observe that the obtained ordering of trip offers is aligned with the intuition. Trip offer 2 has the shortest Trip Duration, comparable waiting time and the smallest number of stops. Consequently, it received the highest score for the offer category Quick. In the offer category Cheap, the highest score is consistently associated with the Trip Offer 1 which features the lowest aggregate price. The top value of offer category Comfortable is associated with Trip offer 1 in all three cases. This is expected as Trip Offer 1 is served by the TSP(s) who achieved the highest number of stars on determinant factors that are associated with this category. Finally, the highest rank of the offer category Environmentally Friendly is associated with Trip Offer 3, where the carbon footprint is the lowest. Hence, the obtained values of offer categories are very well aligned with the intuition.

Table 10 Values of offer categories Quick, Cheap, Comfortable and Environmentally friendly calculated for min-max scores and z-scores and ROC weights. Cells (trip offers) with the top values of scores are highlighted.

Offer category	ROC		
	Trip offer 1	Trip offer 2	Trip offer 3
Quick	0.354	1.000	0.1080
Cheap	1.000	0.307	0.231
Comfortable	1.000	0.153	0.576
Env. Friendly	0.667	0.000	1.000
Offer category	**ROC**		
	Trip offer 1	Trip offer 2	Trip offer 3
Quick	0.725	2.000	0.274
Cheap	1.627	0.265	0.188
Comfortable	1.000	-0.694	0.1528
Env. friendly	0.667	0.000	1.000

4.6. Architecture of the Offer Categorizer

This section provides an overview of the architecture of the offer categorizer and the definition of the role of different types of components composing it.

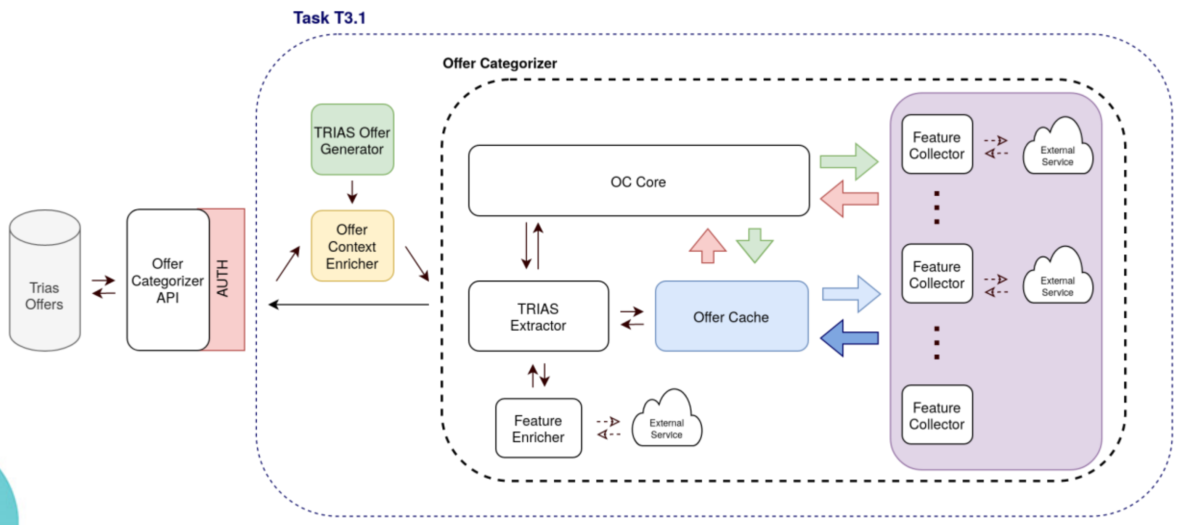


Figure 11 Offer Categorizer Architecture.

The diagram represented in Figure 11 describes the intended integration of the OC within the IP4 ecosystem, however, the architecture of the OC has been designed to be general to make the component easily customisable also within different travel shopping systems. As discussed in Section 3.3.2, the *OfferEnricherAndRanker* component receives a set of travel offers from the *ShoppingOrchestrator* for each request performed by the user. The *OfferEnricherAndRanker* forwards the received travel offers, represented in the TRIAS format discussed in Section 4.3, to the OC which exposes a dedicated *REST API* handling the authorization and authentication of the requests.

One of the goals of the Ride2Rail project is to define additional data about travel offers that a TSP may provide to help a user is choosing among different options. As discussed in the previous sections, the description of offers provided by TSPs is often limited and this can affect the number of determinant factors that can be computed for an offer. For this reason, we introduced in the architecture a component called *Offer Context Enricher* that may be configured to contact additional data sources and/or TSP services to enrich the offers with the information required to compute a higher number of determinant factors. In the development phase, due to the lack of real data, we developed the *TRIAS Offer Generator* mimicking this functionality and generating synthetic data for TRIAS offers as described in Section 4.4. In production, this component is not needed, and additional data would be either provided directly by the TSP in the offer as described in Section 4.3, or gathered by the *Offer Context Enricher*. The possibility of implementing an *Offer Context Enricher* component for the TSPs involved in the Ride2Rail demo sites will be investigated during the project.

After the enrichment of the data, requests are forwarded to the *Offer Categorizer Core* or *OC core*. This is the central component in charge of orchestrating the system and controlling every other piece. Its responsibilities include setting up the connections between the different components, invoking them and combining their outputs.
The first component to be invoked in the categorization process is an *Offer Parser*. We defined as *Offer Parser* a component that is able to parse offers in a given format, extract information and store them in the *Offer Cache* (shared data layer) according to the internal data model defined for the OC. The *Offer Parser*s implemented in Ride2Rail are described in Section 5.4, while the internal data model defined for the *Offer Cache* is described in Section 5.2. An *Offer Parser* may require a *Feature Enricher* subcomponent, possibly connecting to external services, to compute/retrieve information about the offer.
Considering the integration in IP4, the OC core will firstly call the TRIAS extractor component. This module is responsible for parsing the TRIAS offers, converting them to the *Offer Cache* schema and storing the information.

Once this procedure is completed, the *OC core* will call different modules that we named *Feature Collectors*. Each feature collector has the ability to extract the data it needs from the *Offer Cache* and use that from that compute a set of related determinant factors needed for the categorization. For example, all determinant factors related to time (e.g., duration of

the trip, waiting time) are assigned to the same module (Time-FC). Ride2Rail identified and implemented a total number of 9 *Feature Collectors* described in detail in Section 5.5.
We designed *Feature Collectors* as independent microservices to enable the parallelisation of the processing required. Indeed, *Feature Collectors* may require retrieving information from external services (e.g., weather forecast), therefore, it is important to optimise and parallelise the processing.
Once every *Feature Collector* has completed its corresponding computations, it will write the results back to the *Offer Cache* so that the *OC Core* can access them. Finally, accessing these results, the *OC core* will be able to aggregate accordingly the different determinant factors to compute the membership of each offer considering each offer category.

5. OFFER CATEGORIZER DEVELOPMENT

5.1.Software technologies

Building a system such as the OC is challenging in many ways, both in terms of design and implementation. Selecting the correct tools for the development of the different components is vital for a proper and fluid workflow. In this section, we list the main software technologies we use to build the system.

5.1.1.Docker

Docker[17] is an open platform for developing, shipping, and running applications by means of the so-called containers, which are loosely isolated environments. Containers can be understood as a lightweight equivalent of a virtual machine, allowing the users to package an application with all its dependencies separately from the host system.

The most important Docker objects to understand how it works are:
Images: images are the template for building the containers. Often, an image is based on another image, with some additional customization. For T3.1, images are based on the Python image, as well as the configuration details needed to make the applications run.
Containers: containers are the organizational units of Docker. A container is a runnable instance of an image, which essentially means that when an image is built, it runs as a container.

In the project, we dockerized all the different software components – feature collectors, extractors etc. – and we set up a network so that each component can communicate between them.

5.1.2. Flask

Flask[18] is a micro web framework written in Python. It provides you with tools, libraries, and technologies to build web applications. It is a framework with minimal or no dependencies on external libraries.
In our implementation, we use Flask to send the data from the Offer Categorizer core to the different feature collectors. More specifically, the OC core sends the identification associated with the request of user, *request_id*, and with that information, each feature collector can read all the data they need from a cache (see next section). Flask allows us creating an endpoint for the OC core and the feature collectors to communicate.

[17] https://www.docker.com, Docker website, visited on 2021-05-10.
[18] https://flask.palletsprojects.com/, Flask website, visited on 2021-05-10.

5.1.3. Redis

Redis[19], which stands for Remote Dictionary Server, is an open-source, in-memory key-value store, typically used as a database, cache, and message broker. It provides a wide variety of data structures such as strings, hashes, lists or sets. In this Task, we use a Redis Docker image to run an instance and use it as a cache to decrease data access latency, increase throughput, and ease the load off our application. In this way, we can efficiently store the relevant information from the different sources and get specific data using predefined keys.

5.2. Offer Cache

The Offer Cache component described in Section 4.6 represent a shared data layer accessed by the different microservices composing the OC. As discussed in Section 5.1.3, the Redis key-value store was chosen to implement it. To facilitate the integration and the development of the different components, an offer cache schema was defined specifying the set of keys and value datatype used.

We store data in Redis into different levels depending on whether the data is related to the request, the offers, the items or the legs. Each one of the levels can be accessed using a predefined key. For example, to get the data from a specific leg of an offer, the key to be used is:

`<request_id>:<offer_id>:<leg_id>:<data_key>`

where the different ids come from the data sources, and the `<data_key>` varies depending on the data we want to get, e.g., `transportation_mode` to get the mode of transport.

The offer cache schema specifies:

- how the multi-modal offers should be represented to enable the categorization process, i.e., the data format that should be adopted by an *offer parser* to model the multimodal offers received in a specific input data format;
- how the offer features computed by each *feature collector* component are stored;
- how the final categorization of the offers computed by the *OC core* is saved.

The complete offer cache schema is described on Github in the *offer-cache* repository (https://github.com/Ride2Rail/offer-cache).

5.3. Utilities shared across components of the R2R offer categorizer

Repository "**r2r-offer-utils**" contains utilities shared across components of the R2R offer categorizer organized by python scripts:

[19] https://redis.io/, Redis website, visited on 2021-05-10.

- "normalization.py",
- "cache_operations.py",
- "cli_utils.py,
- "logging.py.

5.3.1. Unit "normalization.py"

The Python script ***"normalization.py"*** implements computations used in the quantification of offer categories. It contains the following procedures:
Procedure "zscore()"

```
def zscore(
# a dictionary containing values  of a determinant factor, values are identified by offer identifiers as keys
offers: Mapping,
# binary value indicating whether resulting weights need to be flipped (i.e. substracted from 1)
flipped = False) -> Mapping:
```

The procedure implements the calculation of the z-score weights for a determinant factor across all offers. The algorithm is described in Section 4.5.2. For determinant factors where the higher value of the score is expected to be preferred by passengers (e.g., ticket coverage) the value of parameter *flipped* should be set to "False". In opposite cases, e.g., for price, travel time etc. where it can be expected that in general smaller values are more preferable by passengers the parameter **flipped** should be set to value "True".

Procedure "minmaxscore()"

```
def minmaxscore(
# a dictionary containing values  of a determinant factor, values are identified by offer identifiers as keys
offers: Mapping,
# binary value indicating whether resulting weights need to be flipped (i.e. subtracted from 1)
flipped = False) -> Mapping:
```

The procedure implements calculation of the minmax-score weights for a determinant factor across all offers. . The algorithm is described in Section 4.5.2. For determinant factors where the higher value of the score is expected to be preferred by passengers (e.g. ticket coverage) the value of the parameter *flipped* should be set to "False". In opposite cases, e.g. for price, travel time etc. where it can be expected that in general smaller values are more preferable by passengers the parameter flipped should be set to value "True".

Procedure *"aggregate_a_quantity_over_triplegs"*

```
def aggregate_a_quantity_over_triplegs(
# array of trip leg identifiers to be aggregated
tripleg_ids,
# a dictionary containing weights identified by keys (trip leg key identifiers) that are used as weights
# in the aggregation. Typically, the duration of trip legs is used as a weight.
weights,
# a dictionary containing values of a determinant factor (identified by trip leg identifiers) to be aggregated
quantity)
```

The procedure implements the aggregation of values of a determinant factor for an offer over trip legs belonging to the offer. The algorithm is described in Section 4.5.2.

Unit “cache_operations.py”

The Python script *”cache_operation.py”* implements procedures designed for feature collectors to read and write date from and to the offer cache.

Procedure “extract_data_from_cache()”

```
def extract_data_from_cache(
        # cache identifier
        pa_cache,
        # request id for which the data should be extracted from cache
        pa_request_id,
        # list of attributes at the offer level that should be extracted from the cache
        pa_offer_level_items,
        # list of attributes at the trip leg level that should be extracted from the cache
        pa_tripleg_level_items)
```

A procedure to extract data from the cache. The procedure is applicable inside a feature collector to extract values of selected determinant factors (attributes) at the offer and at the trip leg level. Procedure should not be called directly, but via the wrapper procedure “read_data_from_cache_wrapper()”.

Outputs:

- *output_offer_level_items* - dictionary containing values of requested attributes at the offer level. It contains a list of offer identifiers named "offer_ids" and dictionaries containing values of requested attributes identified by offer id,
- *output_tripleg_level_items* - dictionary containing values of requested attributed at the trip leg level. For each offer id, it contains a list of trip leg identifiers named "triplegs" and dictionaries containing values of requested attributes identified by composite keys.

Procedure “read_data_from_cache_wrapper()”

```
def read_data_from_cache_wrapper(
        # cache identifier
        pa_cache,
        # request id for which the data should be extracted from cache
        pa_request_id,
        # list of attributes at the offer level that should be extracted from the cache
        pa_offer_level_items,
        # list of attributes at the trip leg level that should be extracted from the cache
        pa_tripleg_level_items)
```

Wrapper procedure used by feature collectors to read from cache used by feature collectors. It ensures that repeated reading attempts are made when reading from the cache

Contract No. 881825

fails. After a certain number of unsuccessful attempts, an error is raised. The outputs are the same as those of the **"extract_data_from_cache()"**.

Procedure "store_simple_data_to_cache()"

```
def store_simple_data_to_cache(
    # cache identifier
    pa_cache,
    # request id for which the data should be stored to cache
    pa_request_id,
    # dictionary containing data indexed by offer ids that are supposed to be stored to cache
    pa_data,
    # subkey (final part of the composite key) that is added to request_id and offer_id
    # under which the data is stored in the cache
    pa_sub_key)
```

A procedure to store data produced by a feature collector to the offer cache. The procedure returns value *1* if the writing was successful. The procedure should not be called directly, but via the wrapper procedure "store_simple_data_to_cache_wrapper".

Procedure "store_simple_data_to_cache_wrapper()"
Wrapper procedure used by feature collectors to write data to cache. It ensures that repeated writing attempts are made when writing to the cache fails. After a certain number of unsuccessful attempts, an error is raised. The procedure returns value *1* if the writing was successful.

Unit cli_utils.py
The unit implements a class for integer valued variables. The class is used to ensure a unified control over the feasible range of integer values across different components of OC.

Unit logging.py
The logging of messages about events that happen when the OC runs is ensured by the library logging. The unit implements procedure to set parameters defining the way how the logging of messages is done. Procedure is used across different components of the OC to apply the same configuration for logging.

5.4. Offer Parsers

Offer parsers are responsible of extracting data from an input data format to the internal offer cache schema to enable the categorization process. Ride2Rail has defined two offer parsers, that are discussed in detail in this section, considering the data sources described in Section 4.4.

Contract No. 881825

5.4.1. TRIAS Extractor

The *trias-extractor* offer parser[20] is a module of the Ride2Rail Offer Categorizer responsible for parsing offers from Trias and for converting them to the offer cache schema enabling the categorization process.
The *trias-extractor* expects as input a *Ride2RailScoringRequest* following the IP4 data format discussed in Section 4.3.
The procedure implemented by the *trias-extractor* is composed of two main phases.

Phase I: Parsing

Parsing of data required from the Trias file provided to an intermediate representation using in-memory objects. The procedures to parse the data are implemented in the *extractor* module. The intermediate object model used to represent the parsed data is defined in the *model* module.
The defined model partially reflects the *offer cache* schema:

1. **Request**: id, start_time, end_time, start_point, end_point, offers (dictionary of associated Offer objects)
2. **Offer**: id, trip, bookable_total, complete_total, offer_items (dictionary of associated OfferItem objects)
3. **Trip**: id, duration, start_time, end_time, num_interchanges, length, legs (dictionary of associated TripLeg objects)
4. **OfferItem**: id, name, fares_authority_ref, fares_authority_text, price, leg_ids (list of ids of TripLeg objects covered by the OfferItem object)
5. **TripLeg**: id, start_time, end_time, duration, leg_track, length, leg_stops, transportation_mode, travel_expert, attributes (dictionary of key-value pairs)
    - *TimedLeg*(TripLeg): line, journey
    - ContinuousLeg(TripLeg)
        - *RideSharingLeg*(ContinuousLeg): driver, vehicle

Location and its subclasses (*StopPoint*, *Address*) are used to support the processing but are not serialized in the offer cache.
The parsing procedure is implemented through the following steps:

1. Parse the mobility request data associated with the offers described in the Trias *TripRequest* obtaining a *model.Request* object
2. Parse the *TripResponseContext* associated with the offers described in the Trias *TripResponse* obtaining a list of *model.Location* objects
3. Parse all the Trias *Trips* and the associated *TripLegs* obtaining a set of *model.Trip* objects referencing an ordered list of *model.TripLegs*

[20] https://github.com/Ride2Rail/trias-extractor

4. Parse the Trias *META-Ticket* associated with the different Trias *Trips* obtaining a list of *model.Offer* objects referencing the associated *model.Trip* and bound to the *model.Request* object
5. Parse the Trias *Ticket* associated with each *META-Ticket* obtaining a list of *model.OfferItem* associated with a *model.Offer* and with the *model.TripLegs* covered by the offer item.
6. Parse the *OfferItemContext* for each Trias Ticket obtaining a dictionary of key-value pairs bound to specific *model.TripLegs* associated to the *model.OfferItem*

Additional remarks for the implemented parsing procedure:

- **Step 1**: a UUID is automatically assigned to each request received by the *trias-extractor* and used as id for the *model.Request* object
- **Step 5**: a *model.Offer* can be associated with no *model.OfferItem* if a purchase is not needed to perform the trip
- **Step 6**: If the *OfferItemContext* contains a composite key, the assumption is that it is composed as *oic_key:leg_id* and the parsed value should be associated only with the *model.TripLeg* having the provided *leg_id*. In all the other cases the value parsed is associated to all the *model.TripLegs* associated with the *model.OfferItem*. The information extracted from the *OfferItemContext* is merged with the *Attribute*s parsed for each *model.TripLeg*.

Phase II: Writing

Storing of the data parsed by the *trias-extractor* to the offer cache. A dedicated procedure is defined in the *writer* module. The complete serialization is composed of queued commands in a pipeline that is executed as a single write to the offer cache.

Implementation and usage

The *trias-extractor* component is implemented as a Python application using the *Flask* framework to expose the described procedure as a service accepting HTTP POST requests. Each Trias file processed by the *trias-extractor* component is mapped to a *model.Request* object and then serialized in the offer cache.

Example request running the **trias-extractor** locally:

```
$ curl --header 'Content-Type: application/xml' \
      --request POST  \
      --data-binary '@trias/$FILE_NAME' \
      http://localhost:5000/extract
```

The request_id (key to access the data parsed from the offer cache) is returned in the response as a field in a JSON body together with the number of offers parsed. Example output:

```
{
      "request_id": "581ec560-251e-4dbe-9e52-8f824bda5eb0",
      "num_offers": "15"
}
```

Error code `400` is returned if there is an error in the parsing procedure, code `500` if the request fails for any other reason.

Contract No. 881825

Configuration

The following values of parameters can be defined in the configuration file *trias_extractor_service.conf.*

Section cache:

- *host* - host address of the cache service that should be accessed
- *port* - port number of the cache service that should be accessed

The *trias_extractor/config/codes.csv* can be modified to configure the parsing procedure of the *Attribute*s associated with the different TripLeg nodes and the *OfferItemContext* associated with the different *Ticket* nodes (offer items). The file defines the admissible keys (*key* column), the expected range of the values (*value_min* and *value_max* columns for numeric datatypes) and the datatype (*type* column, admissible values are string, int, float, date) to execute a preliminary validation of the value parsed.

Deployment

Different alternatives are provided to deploy the *trias-extractor* service:

1. Running it **locally** (assumption is that a Redis instance for the Offer Cache is running at `localhost:6379`)
   ```
   $ python3 trias_extractor_service.py
   ```
2. Running on **Docker** (executes both the `trias-extractor` service and a Redis container for the Offer Cache)
   ```
   $ docker-compose build

   $ docker-compose up
   ```
3. A **production deployment** can be obtained changing the `build` section in the docker-compose file to use the `Dockerfile.production` configuration that runs the Flask app on a `gunicorn`[21] server. It is possible to edit the `Dockerfile.production` file to set a different `gunicorn` configuration.
   ```
   $ docker-compose build

   $ docker-compose up
   ```

5.4.2. routeRANK Extractor

[21] https://gunicorn.org/

The routeRANK extractor[22] expects as input the Route Rank data from the MOTIV project described in Section 4.4.2. The data is parsed and stored to the offer cache in the format that is expected by feature collectors and OC core components.

[22] https://github.com/Ride2Rail/routerank-extractor

Contract No. 881825

5.5. Feature Collectors

5.5.1. Weather-fc

The **"weather-fc"** feature collector is a module of the **Ride2Rail OC** responsible for the computation of the ***weather forecast.***

The code implementing **"weather-fc"** feature collector is available on the GitHub Ride2Rail repository: https://github.com/Ride2Rail/weather-fc. The feature collector can be executed by launching the script "***weather.py***", which contains the procedure ***extract()***; the latter is bound under the name ***compute*** with URL using ***FLASK*** (see example request below). This application needs to interact with OpenWeatherMap to get the information required about the weather.

The computation is carried out through the following steps:

1. The request data is extracted from the Redis cache (coordinates and dates of each leg);
2. For each leg, the date and city are stored. Then, we only query the external service for different cities and days (to avoid unnecessary computations). With that, we obtain a classification of each leg into seven different scenarios. Three of them are considered extreme conditions, and they are the ones that could potentially derive in delays.
3. For each leg, we count the number of extreme conditions in the classification and depending on this number we assign a predefined probability of delay (ROD weights).
4. To aggregate the results over all the legs, we just take the higher probability of delay (worst scenario).
5. The absolute scores of the factors are translated into weights through the "***normalization.py"*** script;
6. the factor weights are written into the cache to update the request data, using the "***cache_operations.py"*** script.

Usage

The feature collector can be launched locally by executing the script "***weather-fc.py***" with the following command:

```
$ python3 weather.py
 * Serving Flask app "weather" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 441-842-797
```

Alternatively, the application can be executed as a docker container.

```
docker run \
  --rm \
  -it \
  --name weather \
  -p 5000:5000 \
  --link cache:cache \
  -e FLASK_ENV='development' \
  -v "$PWD":/code \
    r2r/weather-fc:latest
```

Here follows an example of request that can be sent to the application:

```
$ curl --header 'Content-Type: application/json' \
       --request POST  \
       --data '{"request_id": "123x" }' \
         http://localhost:5000/compute
{"request_id": "123x"}%
```

5.5.2. Panoramic-fc

The **"panoramic-fc"** feature collector is a module of the **Ride2Rail OC** responsible for the computation of the following determinant factors: ***"number of monuments", "number of landscapes"*** and ***"number of historical sites".***

The code implementing the **"panoramic-fc"** feature collector is available on the GitHub Ride2Rail repository: https://github.com/Ride2Rail/panoramic-fc. The feature collector can be executed by launching the script "***panoramic.py***", which contains the procedure ***extract()***; the latter is bound under the name ***compute with*** URL using ***FLASK*** (see example request below). This application needs to interact with OpenStreetMap to get the required information.

The computation is carried out through the following steps:

1. The request data is extracted from the Redis cache (coordinates of each leg)
2. For the starting and ending points of each leg, we query the number of monuments, landscapes and historical sites around an area of 100 m radius. The numbers are added up over all the legs to get the total number for each offer.
3. the absolute scores of the factors are translated into weights through the "***normalization.py"*** script;
4. the factor weights are written into the cache to update the request data, using the "***cache_operations.py"*** script.

5.5.3. Price-fc

The **"price-fc"** feature collector is a module of the Ride2Rail OC responsible for the computation of the following determinant factors:
"total_price", ***"ticket_coverage"*** and ***"can_share_cost"***.

The code implementing "price-fc" feature collector is available on the GitHub Ride2Rail repository: https://github.com/Ride2Rail/price-fc . Computation can be executed from the unit ***"price.py"*** by running the procedure ***extract()*** which is bound under the name ***compute*** with URL using ***FLASK*** (see example request below).

Contract No. 881825

Computation is composed of four phases:
Phase I: Extraction of data required by price-fc feature collector from the cache. A dedicated procedure defined" for this purpose from the unit ***"cache_operations.py"*** (repository "r2r-offer-utils) is utilized.
Phase II: Use of the external service to convert prices to EUR (if needed). A class collecting external data has been implemented in the module ***"exchange_rates.py"*** (repository "price-fc").
Phase III: Computation of weights assigned to "price-fc" feature collector. For the aggregation of data at the tripleg level and for normalization of weights dedicated procedures implemented in the unit ***"normalization.py"*** (repository **"r2r-offer-utils"**) are utilized. By default, "z-scores" are used to normalize data.
Phase IV: Storing of the results produced by "price-fc" to cache. A dedicated procedure defined for this purpose in the unit ***"cache_operations.py"*** (repository "r2r-offer-utils) is utilized.

To convert currencies to EUR, python script ***"exchange_rates.py"*** implements class ***exchange_rates*** maintaining information about the exchange rates that are used to convert prices to EUR. The latest information about the exchange rates is downloaded as an xml file from the service[23] provided by the European Central Bank (ECB). More detailed information about the implemented class is available on the Wiki page associated with the repository[24].

Configuration

The following values of parameters can be defined in the configuration file ***"price.conf"***.
Section ***"running"***:

- ***"verbose"*** - if value **"1"** is used, then feature collector is run in the verbose mode,
- ***"scores"*** - if value **"minmax_score"** is used, the min-max approach is used for normalization of weights, otherwise, the **"z-score"** approach is used.

Section ***"cache"***:

- ***"host"*** - host address where the cache service that should be accessed by ***"price-fc"*** feature collector is available,
- ***"port"*** - port number where the cache service that should be accessed used by ***"price-fc"*** feature collector is available.

Example of the configuration file *"price.conf"*:

[23] https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml
[24] https://github.com/Ride2Rail/price-fc/wiki/exchange_rates.py

```
[service]
name = price
type = feature collector
developed_by = Lubos Buzna <https://github.com/lubos31262> and Milan Straka <https://github.com/bioticek>

[running]
verbose = 1
scores  = minmax_scores

[cache]
host = cache
port = 6379
```

Usage

The feature collector ***"price-fc"*** can be launched from the terminal locally by running the script ***"price.py":***

```
$ python3 price.py
 * Serving Flask app "price" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5001/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 441-842-797
```

Moreover, the repository contains also configuration files required to launch the feature collector in Docker from the terminal by the command **docker-compose up**:

```
docker-compose up
Starting price_fc ... done
Attaching to price_fc
price_fc     |  * Serving Flask app "price.py" (lazy loading)
price_fc     |  * Environment: development
price_fc     |  * Debug mode: on
price_fc     |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
price_fc     |  * Restarting with stat
price_fc     |  * Debugger is active!
price_fc     |  * Debugger PIN: 250-384-212
```

To make a request (i.e. to calculate values of determinant factors assigned to the ***"price-fc"*** feature collector for a given mobility request defined by a *request_id*) the command curl can be used:

```
$ curl --header 'Content-Type: application/json' \
       --request POST  \
       --data '{"request_id": "123x" }' \
         http://localhost:5001/compute
```

5.5.4. TSP-fc

The **"tsp-fc"** feature collector is a module of the **Ride2Rail Offer Categorizer** responsible for the computation of the following determinant factors: ***"cleanliness"***, ***"space_available"***, ***"ride_smoothness"***, ***"seating_quality"***, ***"internet_availability"***, ***"plugs_or_charging_points"***, ***"silence_area_presence"***, ***"privacy_level"*** and ***"business_area_presence"***.

Computation can be executed from ***"tsp.py"*** by running the procedure ***extract()*** which is bound under the name ***compute*** with URL using ***FLASK*** (see example request below). The computation is composed of three phases:

Phase I: Extraction of data required by the "**tsp-fc**" feature collector from the cache. A dedicated procedure defined for this purpose from the unit ***"cache_operations.py"*** is utilized.

Phase II: Computation of weights assigned to **"tsp-fc"** feature collector. For the aggregation of data at the tripleg level and for normalization of weights a dedicated procedure implemented in the unit ***"normalization.py"*** are utilized. By default, "z-scores" are used to normalize data.

Phase III: Storing of the results produced by **"tsp-fc"** to cache. A dedicated procedure defined for this purpose in the unit ***"cache_operations.py"*** is utilized.

Configuration

The following values of parameters can be defined in the configuration file ***"tsp.conf"***.

Section ***"running"***:

- ***"verbose"*** - if value **"1"** is used, then feature collector is run in the verbose mode,
- ***"scores"*** - if value **"minmax_score"** is used, the min-max approach is used for normalization of weights, otherwise, the **"z-score"** approach is used.

Section ***"cache"***:

- ***"host"*** - host address where the cache service that should be accessed by ***"tsp-fc"*** feature collector is available,
- ***"port"*** - port number where the cache service that should be accessed used by ***"tsp-fc"*** feature collector is available.

Example of the configuration file ***"tsp.conf"***:

```
[service]
name = tsp
type = feature collector
developed_by = Lubos Buzna <lubos(dot)buzna(at)fri(dot)uniza(dot)sk> and Milan Straka<milan(dot)straka(at)fri

[running]
verbose = 1
scores  = z_scores

[cache]
host = cache
port = 6379
```

Usage

The feature collector **"tsp-fc"** can be launched from the terminal locally by running the script **"tsp.py"**:

```
$ python3 tsp.py
 * Serving Flask app "price" (lazy loading)
 * Environment: development
 * Debug mode: on
```

Moreover, the repository contains also configuration files required to launch the feature collector in Docker from the terminal by the command **docker-compose up**:

```
docker-compose up
Starting tsp_fc ... done
Attaching to tsp_fc
tsp_fc    |  * Serving Flask app "tsp.py" (lazy loading)
tsp_fc    |  * Environment: development
tsp_fc    |  * Debug mode: on
tsp_fc    |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
tsp_fc    |  * Restarting with stat
tsp_fc    |  * Debugger is active!
tsp_fc    |  * Debugger PIN: 248-423-277
```

To make a request (i.e. to calculate values of determinant factors assigned to the "tsp-fc" feature collector for a given mobility request defined by a request_id) the command curl can be used:

```
$ curl --header 'Content-Type: application/json' \
       --request POST  \
       --data '{"request_id": "123x" }' \
         http://localhost:5001/compute
```

5.5.5. Time-fc

The "time-fc" feature collector is a module of the **Ride2Rail OC** responsible for the computation of the following determinant factors: ***"trip duration", "waiting time", "trip to departure", "time of trip (rush vs non-rush hour)".***

The code implementing "time-fc" feature collector is available on the GitHub Ride2Rail repository: https://github.com/Ride2Rail/time-fc. The feature collector can be executed by launching the script "***time-fc.py***", which contains the procedure ***extract()***; the latter is bound under the name ***compute*** with URL using ***FLASK*** (see example request below). This application does not interact with any external services, as it relies solely on the data extracted from the TRIAS request data.

The computation is carried out through the following steps:

1. the request data is extracted from the Redis cache;
2. the determinant factors are computed as described in Section 4.5;

3. the absolute scores of the factors are translated into weights through the "*normalization.py"* script;
4. the factor weights are written into the cache to update the request data, using the "*cache_operations.py"* script.

Usage
The feature collector can be launched locally by executing the script "*time-fc.py*" with the following command:

```
$ python3 time-fc.py
 * Serving Flask app "time-fc.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://172.18.0.3:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 494-689-672
```

Alternatively, the application can be executed as a docker container.

```
docker build -t r2r/time-fc:latest .
```

```
docker run --rm -it --name time \
           -p 5000:5000 \
           --link offer-cache_cache_1:cache \
           -e FLASK_ENV='development' \
           --net cache-network \
           -v "$PWD":/code \
           r2r/time-fc:latest
```

Please note that **"cache-network"** is the network where the offer-cache container runs when launched with docker-compose.
Here follows an example of request that can be sent to the application:

```
$ curl --header 'Content-Type: application/json' \
       --request POST \
       --data '{"request_id": "#31:4265-#24:10239"}' \
       http://localhost:5000/compute
```

Contract No. 881825

5.5.6. Traffic-fc

The **"traffic-fc"** feature collector is a module of the Ride2Rail Offer Categorizer responsible for the computation of the determinant factors: **"traffic"**.
To calculate the **"traffic"** determinant factor, this feature collector relies on the HERE Traffic API[25]. A HERE Developer account is required, and an API Key needs to be inserted in the file **"traffic.conf"**.
Computation can be executed from "traffic.py" by running the procedure **extract()** which is bound under the name compute with URL using FLASK.
Computation is composed of three phases (Phase I, Phase II, and Phase III) in the same way the (https://github.com/Ride2Rail/tsp-fc) use it.
As a categorization of the level of congestion this trip might face **ratio**: Duration with current Flow / Duration with Free Flow. The higher the ratio the worst the traffic condition. The traffic status calculated only in cases where the transport mode must use road network. The categorization ranges from 1-5 while non-road legs get the value 6. The values returned in a normalized format.

5.5.7. Environmental-fc

The **"environmental-fc"** feature collector is a module of the Ride2Rail Offer Categorizer responsible for the computation of the determinant factors: **"total_co2_offer"**, **"co2_per_km_offer"**, **"total_co2_offer_norm"**, **"co2_per_km_offer_norm"**.

- **"total_co2_offer"**, **"total_co2_offer_norm"**: The total gCO2 this offer cause.
- **"co2_per_km_offer"**, **"co2_per_km_offer_norm"**: The total gCO2 this offer cause weighted by the distance of each leg.

To compute those value the trip distance used as independent variable. Namely, that collector assumes that CO2 consumption is a linear function of the distance. Thus, each mode has a CO_2g/km stored in a dictionary.
Computation can be executed from "environmental.py" by running the procedure **extract()** which is bound under the name compute with URL using FLASK. Computation is composed of three phases (Phase I, Phase II, and Phase III) in the same way the (https://github.com/Ride2Rail/tsp-fc) use it.

5.5.8. Position-fc

The "position-fc" feature collector is a module of the Ride2Rail Offer Categorizer responsible for the computation of the determinant factors: **"road_dist"**, **"ratio_dist"**, **"total_stops"**,**"total_legs"**, **"origin"**, **"destination"**, **"road_dist_norm"**, **"total_stops_norm"**, **"total_legs_norm"**, **"ratio_dist_norm"**.

[25] https://developer.here.com

- **"road_dist", "road_dist_norm"**: the total distance performed by transport modes using a road (car, bus, motorbike)
- **"ratio_dist" , "ratio_dist_norm"**: the proportion of distance using road network over the total trip distance.
- **"total_stops", "total_stops_norm"**: the total number of stops this offer have.
- **"total_legs", "total_legs_norm"**: the total number of legs this offer have.

Computation can be executed from **"position.py"** by running the procedure **extract()** which is bound under the name compute with URL using FLASK (see example request below). Computation is composed of three phases (Phase I, Phase II, and Phase III) in the same way the (https://github.com/Ride2Rail/tsp-fc) use it.

5.5.9. Active-fc.

The **"active-fc"** feature collector is a module of the Ride2Rail Offer Categorizer responsible for the computation of the following determinant factors: **"leg_fraction", "bike_walk_distance", "total_walk_distance", "total_distance", and "bike_walk_legs"**.

- **"leg_fraction"**: Calculate the number of legs performed by walk of bike divided by the total number of legs.
- **"bike_walk_distance"**: the distance travelled with bike or walk.
- **"total_walk_distance"**: the distance travelled by walk.
- **"total_distance"**: the total distance of the offer.
- **"bike_walk_legs"**: the number of legs by walk.

That feature collector mostly used in Healthy and Door-to-Door offer Category as walking and bike transport modes are both healthy and flexible for urban trips.
Computation can be executed from "active.py" by running the procedure **extract()** which is bound under the name compute with URL using FLASK (see example request below). Computation is composed of three phases (Phase I, Phase II, and Phase III) in the same way the (https://github.com/Ride2Rail/tsp-fc) use it.

5.6. Offer Categorizer Core

The Offer Categorizer Core **OC-core** is the module that is in charge of the management of the whole workflow of the OC. In particular, its responsibilities include setting up the connections between the other components, combining their input and output, aggregate the determinant factors for each category, carry out the categorization of the offers and handling the errors that might occur at any point in the computation.
From a general perspective, the steps performed by the OC core can be summarized as follows:

1. A TRIAS request is sent to the OC core;
2. The OC core sends the TRIAS data to the **"trias-extractor"** module, which writes the extracted data into the cache and returns the request id to the OC core;
3. The OC core sends a request to all feature collectors, attaching the request id;
4. When all feature collectors respond, the OC core aggregates the determinant factors for each category (according to the algorithm described in Section 5.5).
5. The OC core assigns labels to offers and writes the data into the cache.

Requests are carried out through HTTP protocol, via POST. It is noteworthy that, in step 3, the requests are sent to the different feature collectors in an asynchronous fashion, so as to limit the overall latency to the computation time of the slowest feature collector.

Usage

The service is deployed as a Docker container and can be launched with the `docker-compose up` command, which brings it up together with the Redis cache, the TRIAS-extractor and all the feature collectors. Please note that, in order to be able to execute the application, the user first needs to upload the TRIAS data and initialize the TRIAS-extractor (please refer to Section 5.4.1).

6. CONCLUSIONS

In this deliverable, we have described the results of Task 3.1 (T3.1), as part of Workpackage 3 (WP3) of the RIDE2RAIL (R2R) project. We have introduced a novel system to enrich and classify multimodal travel solutions proposed through the Travel Companion (TC), called the Offer Categorizer.

We built upon the conceptualization provided in D2.4 - "Final Conceptualization of Choice Criteria and Incentives" and we developed a state-of-the-art OC to enable the description of offers along 11 different categories, evaluating each trip offer along several dimensions such as comfort, environmental friendliness, and health impact. This classification enables the implementation of a matching and ranking algorithm, that also learns the user's preferences over time based on their travel choices. Users will thus benefit from the enrichment of travel offers provided by the IP4 ecosystem, and more personalized travel solutions over time.

In practical terms, the OC receives a set of travel offers – i.e., the proposed travel solution for a mobility request made by an IP4 user – in TRIAS format. The first step of the processing performed by the OC is to extract from the TRIAS format the characteristics of each travel offer that are relevant for its classification. Then, a set of Feature Collectors (FCs) compares the offer along a given peculiar characteristic – or Determinant Factor (DF) – that forms the foundation for the computation of a score for a given category.

We highlight that the highly modular design of the OC, based on independent micro-services that run in isolation, follows the best industry practices for software development and should provide the best condition for future enhancements of the system.

All planned objectives of Task T3.1 have been successfully reached and now the OC can be used as a building block for the development of the other planned software components in WP3 and for the execution of the R2R demos. Furthermore, the partners involved in the task have constantly provided updates on the development so that there should be no constraints for the development of the software components coming from the design and implementation of the OC.

The code produced and described in this task is publicly available on GitHub under the RIDE2RAIL organization at https://github.com/Ride2Rail. This deliverable is intended to be a companion to the software documentation that we have provided for every component, and we refer to the code repositories for the latest and most updated information.

Contract No. 881825

7. REFERENCES

RIDE2RAIL. (2020). *Deliverable D2.1 - First Conceptualization of Choice Criteria and Incentives.* Retrieved from https://ride2rail.eu/resources-library/

RIDE2RAIL. (2020). *Deliverable D2.4 - Final Conceptualization of Choice Criteria and Incentives.* Retrieved from https://ride2rail.eu/resources-library/

RIDE2RAIL. (2020). *Deliverable D2.6 - Final Set of Requirements and Specification for Complementary Travel Expert Services.* Retrieved from https://ride2rail.eu/resources-library/

RIDE2RAIL. (2020). *Grant agreement number 881825.*

Roberts, R., & Goodwin, P. (2002). Weight approximations in multi-attribute decision models. *Journal of Multi-Criteria Decision Analysis, 11*(6), 291–303. https://doi.org/10.1002/mcda.320

8. APPENDICES

APPENDIX A: Offer Cache with the MoTiV/routeRANK data fields

In this Appendix, we discuss the mapping between the Offer Cache and MoTiV/RouteRANK data. As mentioned in Section 4.4.2 RouteRANK data are originally available in JSON format, while the datacoming from the S2R are available in TRIAS format. To use these data during the development of the OC, we have established a mapping between the Offer Cache fields and the RouteRANK data.

Offer-level information

Prefix <request_id>:<offer_id>

- duration

Duration of the trip associated to the offer.
Obtained from `.alternatives[].segments.duration` - espressed in seconds - and converted to a `String` formatted as `xsd:duration`

- start_time

Starting time of the trip associated to the offer.
Obtained combining .alternatives[].segments[].departureDate and .alternatives[].segments[].departureDate and converting them to a String formatted as xsd:datetime

- end_time

Ending time of the trip associated to the offer
Obtained combining.alternatives[].segments[].arrivalDate and .alternatives[].segments[].arrivalTime and converting them to a String formatted as xsd:datetime

- num_interchanges

Number of interchanges of the Trip associated to the offer.
An `Integer` obtained as the number of legs minus 1.

- legs

A list of `<leg_id>` composing the trip associated to the offer.
Obtained from `.alternatives[].segments[].legs[].id` and converted to a `List` of `Strings` representing `<leg_id>` keys

- offer_items

List of `<offer_id>` composing the offer.
Not available from the RouteRANK data.

Contract No. 881825

List of Strings representing `<offer_id>` keys

- bookable_total

Total of all offer items that are actually bookable.
Not available from the RouteRANK data.
`Integer` (last two digits are decimal, e.g., 6215 EUR represents 62,15 euro)

- complete_total

Total of all offer items.
Obtained from `.alternatives[].segment[].totals.price` and converted to a `List` of `Integer` (last two digits are decimal, e.g., 6215 EUR represents 62,15 euro)

- currency

Currency for the reported totals.
Always set to `EUR`.
`Integer` formatted as ISO4217 currency code, e.g., EUR for Euro

Leg-level information
Prefix <request_id>:<offer_id>:<leg_id>

- leg_type

Type of leg.
Obtained from transport mode:
  - `walking`, `bike`, and `car` legs are set to ridesharing leg
  - `train`, `taxi`, `change`, `bus`, `subway`, `tram`, `genericpubtrans`, `boat`, and `funicular` legs are set to timed leg
  - `carsharing`, and `bikesharing` legs are set to ridesharing leg

`String`. Admissible values are: `timed`, `continuous`, `ridesharing`

- start_time

Starting time of the Leg associated to the offer.
Obtained combining `.alternatives[].segments[].departureDate` and `.alternatives[].segments[].departureDate` and converting them to a `String` formatted a
`String` formatted as `xsd:datetime`

- end_time

Ending time of the Leg associated to the offer.
Obtained combining `.alternatives[].segments[].arrivalDate` and `.alternatives[].segments[].arrivalTime` and converting them to a `String` formatted as `xsd:datetime`

- duration

Duration of the Leg associated to the offer.

Contract No. 881825

Obtained as `start_time - end_time` and converted to a `String` formatted as `xsd:duration`. If continuous (or ridesharing) can be different from (`end_time - start_time`) because a time window to perform the leg can be defined.

- transportation_mode

Transportation Mode for the leg.
A `String` obtained as follows:

  - Modes in RouteRANK not in the specifications: `bikesharing, change, genericpublictrans`
  - Modes in the specifications not in RouteRANK: `all, unknown, trolleyBus, coach, intercityRail, urbanRail, cableway, motorcycle, truck`
  - Mapping names from RouteRANK to cache:
    - `bike, bikesharing: cycle`
    - `subway: metro`
    - `carsharing: others-drive-car`
    - `train, genericpublictrans: rail`
    - `car: self-drive-car`
    - `walking: walk`
    - `boat: water`

Admissible values are:

  - timed legs: all, unknown, air, bus, trolleyBus, tram, coach, rail, intercityRail, urbanRail, metro, water, cableway, funicular, taxi
  - continuous legs: walk, cycle, taxi, self-drive-car, motorcycle, truck
  - ridesharing legs: others-drive-car

- leg_stops

Coordinates of the stops performed during the leg.
`String` formatted as GeoJson `MultiPoint`. At least coordinates of starting and ending point.

- leg_track

Coordinates of the track followed during the leg.
`String` formatted as GeoJson `MultiPoint`.

- travel_expert

Identifier of the Travel Expert associated to the leg.
Not available from the RouteRANK data.
`String`

- line

Identifier of the Line for the service of a Timed Leg.
Not available from the RouteRANK data.
`String`

Contract No. 881825

- journey

Identifier of the Service Journey for the service of a Timed Leg
Available only for timed legs.
Not available from the RouteRANK data.
`String`

- driver [only for ridesharing legs]

Identifier of the Driver for the service of a Ridesharing Leg.
Available only for timed legs.
Not available from the RouteRANK data.
`String`

- passenger [only for ridesharing legs]

Identifier of the Passenger for the service of a Ridesharing Leg
Available only for timed legs.
Not available from the RouteRANK data.
`String`

- vehicle [only for ridesharing legs]

Identifier of the Vehicle for the service of a Ridesharing Leg.
Available only for timed legs.
Not available from the RouteRANK data.
`String`

Offer-Item-level information
Prefix <request_id>:<offer_id>:<item_id>

- price

Price for the offer item
Obtained from `.alternatives[].segment[].totals.price` and converted to a `List` of `Integer` (last two digits are decimal, e.g., 6215 EUR represents 62,15 euro)

- currency

Currency for the reported price.
Always set to `EUR`.
`String` formatted as ISO4217 currency code, e.g. EUR for Euro

- name

Printable ticket name.
Not available from the RouteRANK data.
`String`

- fares_authority_ref

Reference to a Fares Authority defining the price for the offer item.
Not available from the RouteRANK data.

Contract No. 881825

`String`

- fares_authority_text

Textual description or name of Fares Authority defining the price for the offer item.
Not available from the RouteRANK data.
`String`

- legs

List of `<leg_id>` covered by this offer item
List of strings representing `<leg_id>` keys