BONVOYAGE

From Bilbao to Oslo, intermodal mobility solutions, interfaces and applications for people and goods, supported by an innovative communication network

Research and Innovation Action GA 635867

Deliverable D5.1:

Design of the adaptation functionality

Deliverable Type:	Report
Deliverable Number:	5.1
Contractual Date of Delivery to the EU:	28.02.2017
Actual Date of Delivery to the EU:	07.03.2017
Title of Deliverable:	Design of the adaptation functionality
Work package contributing to the Deliverable:	WP5
Dissemination Level:	PU
Editor:	Dag Kjenstad [SINTEF]
Author(s):	Dag Kjenstad, Lukas Bach, Carlo Mannino [SINTEF], Ignacio González Fernandez, Jordi

Fonoll [ATOS], Stephan Strodl [FLUIDTIME], Giuseppe Tropea, Giuseppe Piro, Pietro Boccadoro, Giuseppe Ribezzo, Giulio Rossi, Andrea Detti, Nicola Blefari Melazzi [CNIT]

Internal Reviewer(s): Andrea Detti [CNIT]

Abstract: This document reports the design of the adaptation functionality of the BONVOYAGE mobility platform. The design has particular attention to the main logic modules, i.e., the Multi-Modal Mobility Database, the Discovery Service and Metadata Handling Tool, and the adaptation of the trip planning services and internal interfaces.

Keyword List: Adaptation functionality, distribution of spatial data, federated trip planning services, intelligent transportation system, standardized travel data.

TABLE OF CONTENTS

List of Figures

List of Tables

BONVOYAGE Glossary

In Table 1 we have listed and described the terms that have been considered central and relevant in this deliverable.

BONVOYAGE GLOSSARY	
TERM	**DEFINITION**
Adapt	To change something so that it functions better or is better suited for a purpose.
Adaptation Functionality	Functionalities for the external services to be able to communicate with the BONVOYAGE platform.
DATEX II	The standard format for exchange of traffic management information developed in line with the ITS Action Plan.
GTFS	General Transit Feed Specification (GTFS) is an open standard data format for exchange of public transport timetables used by many applications and transport agencies.
Metadata Handling Tool	A component of the BONVOYAGE architecture
Multi-Modal Mobility Data-base	A component of the BONVOYAGE architecture
NeTEx	NeTEx is the emerging CEN Technical standard for exchanging public transport information as XML documents.
Orchestrator	The BONVOYAGE Orchestrator is a decomposition approach to solve the trip planning on a multimodal network by means of soloists. Orchestrators can act as national access points for trip planning.
Overlay graph	An aggregated, high-level graph used by the BONVOYAGE Orchestrator to organ-ize single soloists into a coherent federation of coordinated trip planning ser-vices.
Soloist	The soloists are distributed trip planning services to handle trip planning tasks for a part of the overall transportation and road network
SPROUTE	SPROUTE is an open source format to exchange routing information (request and response) as JSON objects.

Table 1: BONVOYAGE Glossary

1 Introduction

1.1 Deliverable Rationale

The biggest challenge WP5 had to face during the first half of the project's life was to **align its design work with the innovative directions dictated by the WP2 architecture**. Specifically the Intelligent Transport Functionality, the Multi Modal Mobility Database and the Meta Data Handling Tool were initially conceived as logically centralized components, but subsequent developments of the project (see D2.2 and D1.2) affirmed the design of BONVOYAGE as a **continent-wide federated platform of collaborative but independent travel planning services and data-sources**. Such architecture allows for a truly innovative model in the scenario of journey planners, which is a distinctive feature of our project (see D1.2). Differently than a generically distributed or P2P system, a federation is a group of computing or network providers agreeing upon standards of operation in a collective fashion, so that inter-operation of such group of distinct, formally disconnected networks, which have different internal structures, is obtained. To establish the infrastructure for continent wide travel planning, we have grounded the federation on an information-centric network layer (see D3.1).

This deliverable describes how the design of the Multi Modal Mobility Database, of the Meta Data Handling Tool and of the Service Adaptation components has been carried out and refined in order to adhere to the challenges of a federated architecture.

1.2 Quality review

VERSION CONTROL TABLE			
VERSION NO.	PURPOSE/CHANGES	AUTHOR	DATE
0.1	FRONT PAGE AND TENTATIVE TOC	DAG KJENSTAD	01/12/2016
0.2	ASSEMBLED WITH PARTNER CONTRIBUTIONS	ALL PARTNERS	21/01/2017
0.5	FIRST FULL TEXT REVISION	GIUSEPPE TROPEA	15/02/2017
0.6	REVISED SINTEF CONTRIBUTIONS	LUKAS BACH	16/02/2017
0.7	REVISED CNIT CONTRIBUTIONS	GIUSEPPE PIRO	18/02/2017
0.8	REVISED ATOS CONTRIBUTIONS	JORDI FONOLL	19/02/2107
0.9	NOVEL SINTEF CONTRIBUTIONS	DAG KJENSTAD	27/02/2017
0.9	SECOND FULL TEXT REVISION	GIUSEPPE TROPEA	28/02/2017
0.95	INTERNAL REVIEW	ANDREA DETTI	28/02/2017
1.0	AUTHORIZED	ANDREA DETTI	06/03/2017

Please notice that for this deliverable the contractual date has been officially moved to February 28, 2017.

1.3 Executive summary

1.3.1 Deliverable description

Work Package 5 is about Adaptation Functionality needed for external and internal services to interwork each other in the BONVOYAGE platform. We identified three main functionality, namely: the Multi-Modal Mobility Data Base (Chapter 2), Discovery services (Chapter 3) and Adaptation services (Chapters 4 and 5), whose design is reported in this deliverable.

It is worth to remind that in Deliverable D1.2 we presented the BONVOYAGE architecture as formed by i) a federated architecture providing discovery services, planning services (aka soloists or solvers) and data sources; ii) an orchestrator that discovers and coordinates the use of planning services, in order to carry out multi-objective travel optimizations; iii) and an application server that serves end-users by contacting the orchestrator.

In this framework, the Modal Mobility Database (MMMDB) is a logical component of the BONVOYAGE architecture that collects all the information necessary to derive an optimal door-to-door intermodal trip plan for a given user. The MMMDB stores both user related information and also high-level topological information necessary to carry out a multi-objective travel optimization. The BONVOYAGE architecture motivates a distributed design of the MMMDB, splitting it in to the Application Server, for what concerns the User Data Model (WP4), and the Orchestrator (WP4), for what concerns the topological information model (aka orchestrator graph).

Orchestrators need to discover planning services (e.g. carpooling in a city or train service in a country) and, in turn, planning services need to discover ITS data sources. Consequently a Discovery service is a central component of the BONVOYAGE architecture. It is implemented by a federated spatial database named OpenGeoBase (WP3), supported by a Metadata Handling Tool (MDHT) that makes the different planning services and data sources (GTFS, DATEXII, etc.) discoverable through OpenGeoBase. Of particular importance to the design of the discovery service has been the representation of ITS information regarding relevant standards and the services and data available from the core cities participating in the BONVOYAGE project, Bilbao, Oslo and Rome, especially dealing with real-time feeds.

Lastly, we observe that many components of the BONVOYAGE architecture need to be adapted in order to expose them “as-a-service”. Actually these are the orchestrator, the planning services and sources of real time travel-data.

1.3.2 Summary of results

We achieve the design of a Multi Modal Mobility Database split into a User MMMDB and an Orchestrator MMMDB.

We achieve a set of extensions to the SPROUTE format, enabling the inter-linking of planning services with an Orchestrator, and the inter-linking of the Orchestrator with Application Servers.

We achieve the design of a Discovery service based on a federated spatial database, and on a metadata extracting tool, which is able to index large-scale world-wide resources, namely data and planning services.

We achieve the integration of the Discovery Service with an Information-Centric Networking layer (the BONVOYAGE Communication System, see D3.1) able to natively support Publish/Subscribe interaction between producers and

consumers of information. This integration enables BONVOYAGE to cope with highly dynamic feeds on travel data, achieving efficient and incremental dissemination of real-time data to end-users.

2 Design of the Multi-Modal Mobility Database

The Multi-Modal Mobility Database (MMMDB) is central to BONVOYAGE. It is the component that supports the following functionalities of the Reference Architecture (see D2.2):

Intelligent Transport Functionality

- Planning and travel itinerary management
- Travel objective and target management
- Travel solution management
- User feedback and profile management

Adaptation Functionality

- Geo-location service
- Passenger, freight and travel management
- Profile and account management

Although the MMMDB was initially conceived as a logically centralized component, subsequent developments of the project (see D2.2 and D1.2) motivate its split into a **User MMMDB** and an **Orchestrator MMMDB**. The User MMMDB is built upon the concept of a User Data Model that deals with user-related data, which is kept in a logically centralized Application Server (again, see D1.2 for a better visualization of the BONVOYAGE layers and the role of the Application Server). The Orchestrator MMMDB is an information system within the Orchestrator, which is agnostic of any user-related information and is able to scale efficiently by organizing different independent solvers into a coordinated entity. It deals with mobility-related data and maintains a coordinated high-level graph of solvers that is used to compute the overall travel solutions spanning multiple soloists (and, consequently, multiple modalities and geographic areas).

The following sections of this chapter present the “design” of the User MMMDB, i.e. of the User Data Model. Chapter 4 presents instead the design of the Orchestrator MMMDB, i.e. the high-level graph maintained by the Orchestrator and the Soloists and how they are operated as inter-linked services able to respond to requests for travel plans. We point out that D4.1 reports a more algorithmic description of the Orchestrator+Soloist concept and the actual implementation and deployment of MMMDB will be carried out in WP7.

2.1 Design of the User Data Model

According to American National Standards Institute (ANSI), there exist three kinds of data-model instances:

- Conceptual schema: describes the semantics of a domain (the scope of the model). For example, it may be a model of the interest area of an organization or of an industry. This consists of entity classes, representing kinds of things of significance in the domain, and relationships assertions about associations between pairs of entity classes. A conceptual schema specifies the kinds of facts or propositions that can be expressed using the model. In that sense, it defines the allowed expressions in an artificial "language" with a scope that is limited by the scope of the model. Simply described, a conceptual schema is the first step in organizing the data requirements.

- Logical schema: describes the structure of some domain of information. This consists of descriptions of (for example) tables, columns, object-oriented classes, and XML tags. The logical schema and conceptual schema are sometimes implemented as one and the same.
- Physical schema: describes the physical means used to store data. This is concerned with partitions, CPUs, table-spaces, etc.

This approach allows the three perspectives to be relatively independent of each other. Storage technology can change without affecting either the logical or the conceptual schema. The table/column structure can change without (necessarily) affecting the conceptual schema. In each case, of course, the structures must remain consistent across all schemas of the same data model.

While progressing from requirements to the implemented User Data Model information system, as a first step, requirements have been translated into a conceptual data model, which is essentially a set of technology independent specifications about the data, which has been also used to discuss requirements with the business stakeholders.

As a second step, the conceptual model is then translated into a set of logical entities, which describe how the conceptual structures can be implemented in databases. Implementation of one conceptual data model may require multiple logical data models.

The last step in data modelling would be transforming the logical data model to a physical data model that organizes the data into tables, and accounts for access, performance and storage details. Data modelling defines not just data elements, but also their structures and the relationships between them.

The following section 2.1.1 documents the first step of the design, while section 2.1.2 documents the second step of the design. Upcoming deliverable D5.2 will document the last step.

2.1.1 User Data Model conceptual schema

In this section each component of the User Data Model is defined and presented. The model has been conceived as the summary of all data objects allowing methods and algorithms for service personalization, trip planning and monitoring and other ancillary service to work properly. The User Data Model described in this section has been shared with partners involved in WP4, with particular attention to Task 4.1 User profiler tool, Task 4.2 Multi-objective optimization tool and Task 4.3 Tariff scheme design tool. The aim is that of yielding a reference specification document to the conceptual and logic design of the Multi-Modal Mobility Database (Task 5.2).

The User Data Model has been developed in order to design a customize system for the user's specific needs.

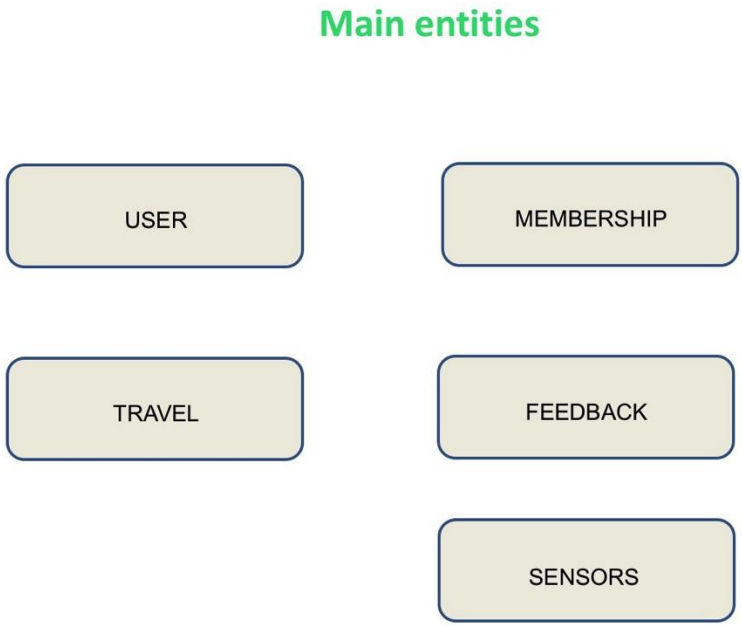

Figure 1: User Data Model - Main entities

User Data Model is a logical representation of entities and relationships between two (or more) entities, describing the user from the personalization point of view. Basically, we considered five main entities (i.e., a tangible and physical object, for instance, human being or in this case a User):

- *User*: this entity represents the user in BONVOYAGE.
- *Travel*: this entity represents the travels characteristics.
- *Membership*: this entity represents which are the group or groups of which the user is member.
- *Feedback*: this entity contains the feedback left by each user about the service provided by BONVOYAGE.
- *Sensors*: available sensors useful to acquire implicit feedbacks.

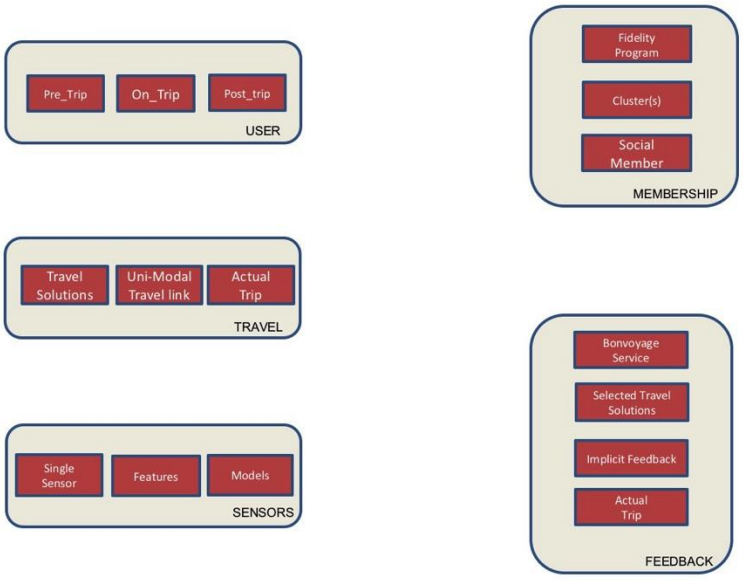

Figure 2: User Data Model - Main sub-entities

The entities, previously introduced, might contain several different sub-entities, (Figure 2 summarizes the sub-entities, and they are sketched in red).

In particular, the main entity *User* contains three different sub-entities:

A) *Pre_trip*: is the sub-entity that represents the user before a Travel

B) *On_trip*: is the sub-entity that represents the user during the Travel

C) *Post_Trip*: is the sub-entity that represents the user after he made a Travel

The main entity *Travel* contains three sub-entities:

A) *Travel Solutions*: is the sub-entity that represents all the travel solutions, ranked for a user, when it requests a travel

B) *Unimodal Travel link*: is the sub-entity that represent each single path into the overall travel

C) *Actual Trip*: is the sub-entity that represent the overall travel composed by at least one unimodal travel link

The main entity *Feedback* contains four sub-entities:

A) *BONVOYAGE Service*: A feedback referred to the overall BONVOYAGE service.

B) *Selected Travel Solutions*: A feedback for the travel solutions proposed by the system to each user.

C) *Implicit Feedback*: A feedback directly acquired by sensors

D) *Actual Trip*: A feedback for the overall travel experience, the travel evaluation.

The main entity *Membership* contains three sub-entities:

A) *Fidelity program*: specification about fidelity programs (for example a fidelity card, etc.)

B) *Cluster(s)*: one or more cluster/s of which each user belongs to.

C) *Social Member*: specification about which social account each user used (e.g., from Google+ or Facebook) from which we are in charge to take information

The main entity *Sensors* contains three sub-entities:

A) *Single sensor*: sensors consists of several "single sensor"

B) *Features* are relevant information computed from the different sensors

C) *Models* are functions that estimate User Stress Level and User Transportation Mode from features, using machine learning algorithms and techniques.

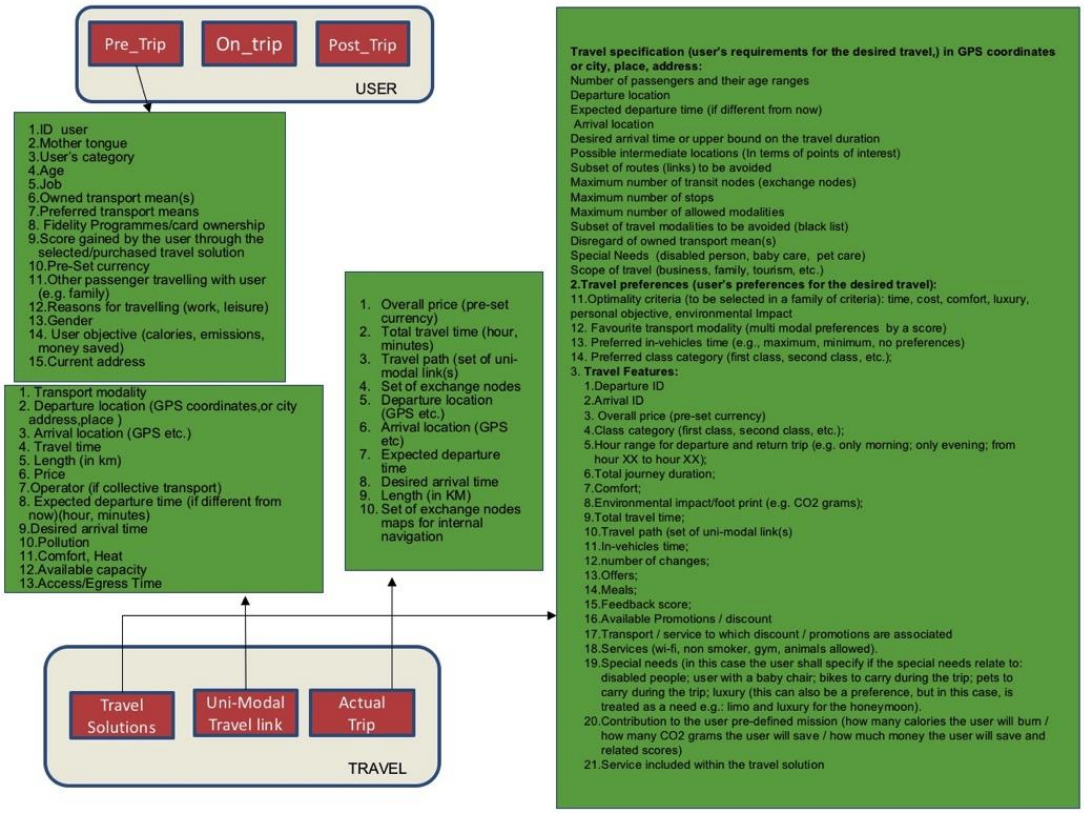

Figure 3: User Data Model - USER and TRAVEL - Main attributes

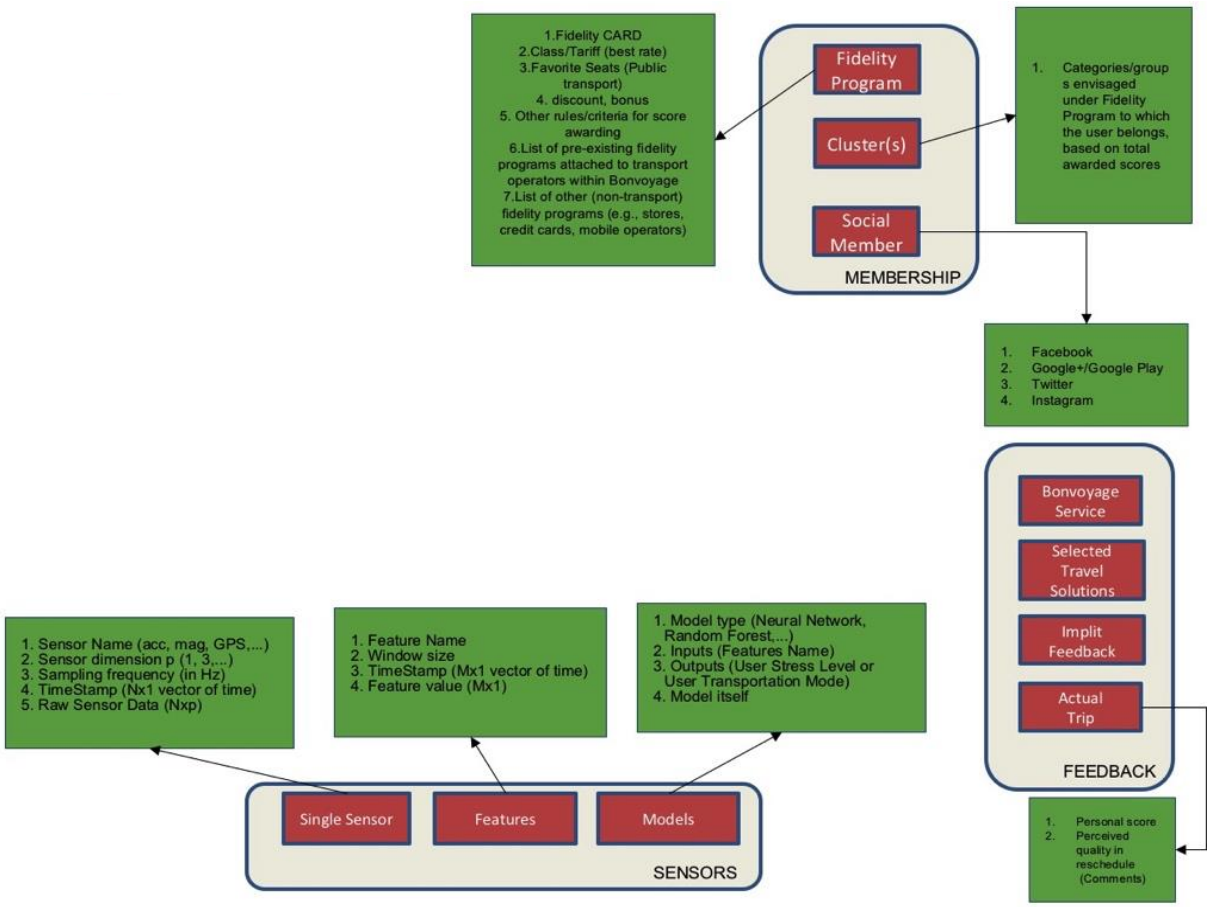

Figure 4: User Data Model – SENSORS, MEMBERSHIP and FEEDBACK - Main attributes

We consider several different attributes for the sub-entity and relations, the attributes are showed in green.

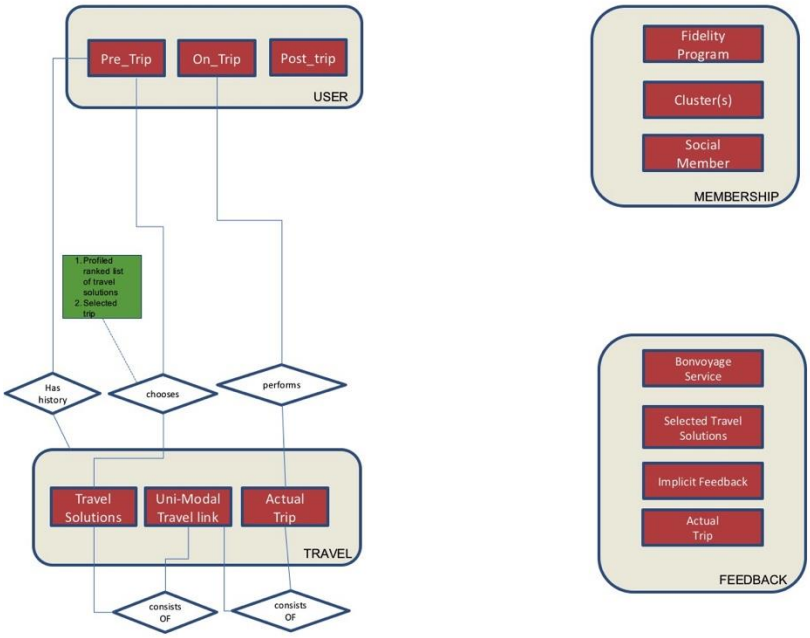

Figure 5: User Data Model - USER and TRAVEL - Main relations

Each sub-entity can have a relation with other sub-entities, the rhomboidal shapes represent the relations between each sub-entity with each other.

For instance, each user in the *Pre_trip* time step chooses a *Travel Solutions* among the other, the user after its last choice become an *On_trip* user, in the *On_trip* time step the user performs an *Actual Trip*, where the *Actual Trip* consists of several different *Unimodal Travel link*.

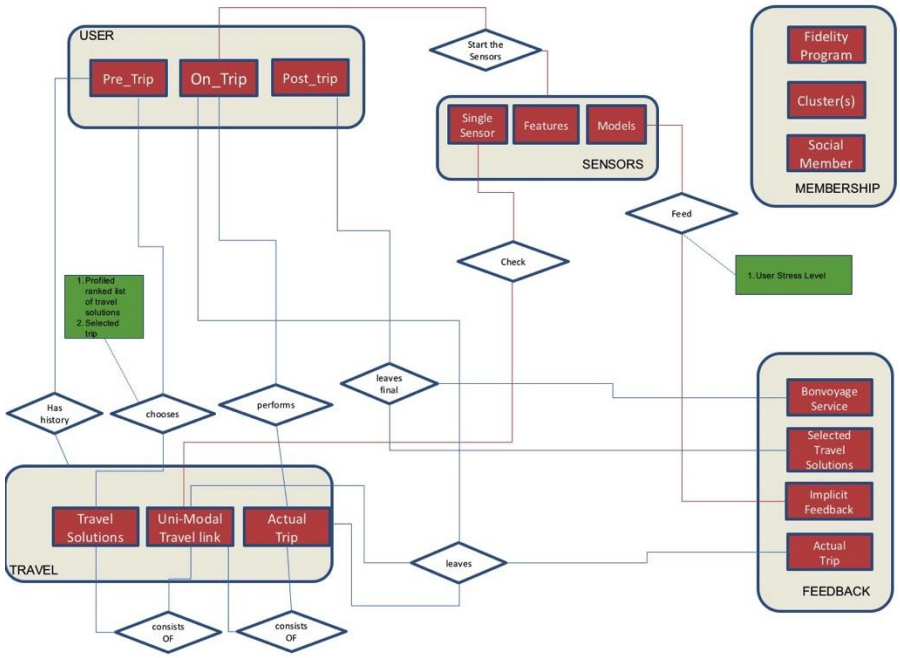

Figure 6: User Data Model - TRAVEL, SENSORS and MEMBERSHIP - Main attributes

Each feedback can be left in the *On_trip* time step and at the end of the travel in the *Post_trip* time step for:

- Travel Solutions ranked (if it is pertinent to the user's preferences), and for the BONVOYAGE Service (for example a feedback for the BONVOYAGE App).

The *Sensors* Entity:

- The sensors are started when each user passes from *Pre-trip* to *On_trip*
- *User Stress Level* is an output of the sub-entity *Models*, and it is an attribute of the relationship *Feed* between *Models* and *Implicit Feedback*.
- *Sensors* check the *Unimodal Travel link*: due to real time GPS position, it is possible to check that the *Actual Trip* which corresponds to the chosen travel solution.

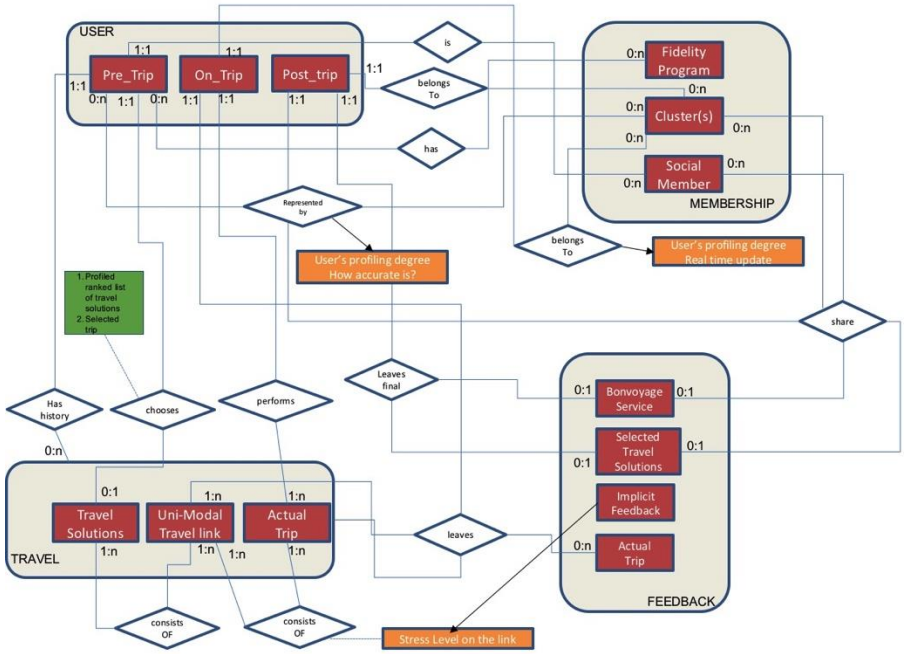

Figure 7: User Data Model - Focus on explicit and implicit user feedback

We figure out the possibility to share each feedback, since it can be a useful guidance for other users that are interested in similar travels, for this reason each feedback can become social thought the relation *share*, such that it can be shared:

- With the other BONVOYAGE users
- With users in the social members
- With users in the same cluster or clusters

The relations between User and Membership are characterized by:

- A *Pre_Trip User* is represented by: one or more clusters. It may have a fidelity program, and it is a social member.
- An *On_Trip User* can belong to a new cluster during the trip.
- A *Post_Trip User*, will update, eventually in a new cluster.

2.1.2 User Data Model logical schema

The preliminary schema presented through the entities and sub-entities (Figure 1 trough Figure 7) has been further elaborated and a more concise and structured model has been developed. Basically, we still considered the set of five main entities with related sub entities. Once refined the data model, it has been translated in an ER logic model in order to formalize the data objects to develop the database, where the data provided by user and algorithms will be stored.

In the following Tables, the main and most representative logical entities of the data model will be described in terms of the keys that correlate them.

Logical entity	REGISTERED_USER
Description	Main information of the user of the BONVOYAGE platform
Relations	MEANS SCORE_POLICY USER_QUERY TRANSPORT_MEAN TRAVEL_SOLUTION CAR_SHARING_RESERVATION FIDELITY_PROGRAM USER_PROFILE

Logical entity	MEANS
Description	Means preferred by a single BONVOYAGE platform registered user
Relations	REGISTERED_USER

Logical entity	SCORE_POLICY
Description	Table needed for the score policy algorithm
Relations	REGISTERED_USER

Logical entity	USER_QUERY
Description	Data managed by the travel data queries
Relations	REGISTERED_USER TRANSFER USER_CLASS OPTIMALITY_CRITERION TRANSPORT_MEAN

	TRAVEL_NEED USER_PROFILE

Logical entity	TRAVEL_NEED
Description	Travel need associated to a query
Relations	USER_QUERY

Logical entity	TRANSPORT_MEAN
Description	Characteristics of a single transport mean
Relations	REGISTERED_USER TRAVEL_NEED USER_QUERY TRANSPORT_SERVICE

Logical entity	USER_CLASS
Description	Categorization of different user groups
Relations	USER_QUERY USER_PROFILE OPTIMALITY_CRITERION

Logical entity	USER_PROFILE
Description	Features of the profile of a single registered user
Relations	USER_CLASS USER_QUERY REGISTERED_USER

Logical entity	FIDELITY_PROGRAM

Description	Tags linked with the fidelity program requirements and features
Relations	REGISTERED_USER TRANSPORT_OPERATOR

Logical entity	TRAVEL_SOLUTION
Description	Transport option chosen by the user
Relations	REGISTERED_USER TARIFF_SCHEME ANCILLARY_SERVICE USER_QUERY TRAVEL_LINK ACTUAL_TRIP

Logical entity	TARIFF_SCHEME
Description	Tariff information
Relations	TRAVEL_SOLUTION TRANSPORT_OPERATOR

Logical entity	ANCILLARY_SERVICE
Description	Special services offered by transport operators
Relations	TRAVEL_SOLUTION TRANSPORT_OPERATOR

Logical entity	CAR_SHARING_RESERVATION
Description	Shared car appointment data
Relations	SHARED_CAR TRANSFER

	REGISTERED_USER

Logical entity	TRANSFER
Description	Shared car transfer info
Relations	CAR_SHARING_RESERVATION POOL

Logical entity	POOL
Description	Car pool linked to a transfer item
Relations	TRANSFER

Logical entity	SHARED_CAR
Description	Real time information of a shared car
Relations	CAR_SHARING_RESERVATION CAR

Logical entity	CAR
Description	Individual car information
Relations	SHARED_CAR CAR_SHARING_OPERATOR

Logical entity	CAR_SHARING_OPERATOR
Description	Information about the car sharing company
Relations	CAR

Logical entity	TRAVEL_LINK
Description	Node between different travel segments

Relations	TRANSPORT_OPERATOR TRANSPORT_SERVICE TRANSPORT_MEAN TRAVEL_NODE

The following Figure shows the ER logic schema where data objects with their relative attributes are described. It can be divided into six macro data objects:

A) *User* includes the entities that characterize the BONVOYAGE user (*Registered User*, *User query*, *User Class*, *User Profile*);
B) *Travel* describes the travel features (*Travel Solutions*, *Travel Link*, *Travel Nodes*, *Tariff Scheme*);
C) *Car Pooling* includes the entities that rely on the carpooling services (*Transfer*, *Pool*, *Shared Car*, *Car*);
D) *Operator* includes the entities that describe the Transport Operator (*Transport Operators*, *Transport Service*, *Ancillary Services*);
E) *Feedback* includes the entities characterizing users' feedbacks (*Feedback Actual Trip*, *Feedback App*);
F) *Sensor* describes the entities capturing sensors activities. Behavioural data include data from smartphone sensors (accelerometer, magnetometer, gyrometer, GPS, used to recognize the transport mode) as well as from Empatica© wristband sensors, used to assess the user's stress level by providing cardiac activity, electrodermal activity and skin temperature. The output is a timestamp plus transportation mode (string array) or user's stress level (digit - between 0 and 1 – array).

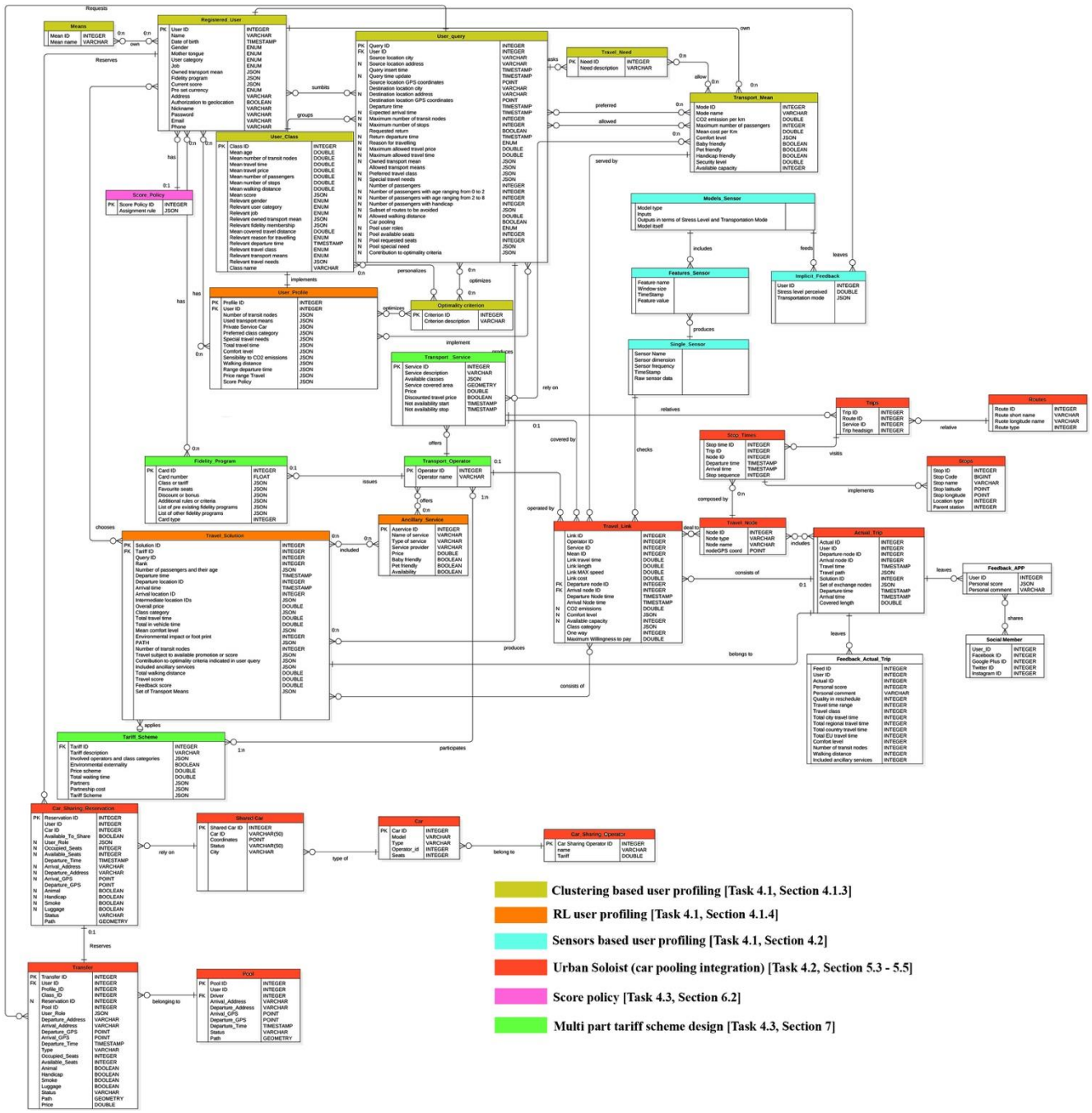

Figure 8: ER logic Schema

The interaction between the BONVOYAGE Intelligent Functionalities and the data objects depicted in the above Figure is fully described in Deliverable D4.1.

3 Design of the discovery service

Besides user-related data, which is kept into the User MMMDB, BONVOYAGE is able to deal with large sets of travel-centric data, both static and dynamic, which may come from diverse and independent business and administrative domains and with a set of equally independent planning services, called Soloists. As explained in D2.2 and D1.2 we achieve this by means of a federation of data sources and services, which we can collectively refer to as “resources”.

Key to federation is the idea that data stays at the producer, which keeps control of its data and is in charge of updating it, while **BONVOYAGE performs indexing of the available resources by means of a distributed spatial database able to accommodate the federated nature** of such travel-data/service producers.

The following sections of this chapter explain how indexing and discovery of federated resources is performed in BONVOYAGE.

3.1 Representation of resources for discovery

OpenGeoBase (OGB) is a NoSQL spatial database we have developed in BONVOYAGE to allow any provider of travel resources (data or soloists) to share them within a federation that is easy to join and allows cooperation and grouping of resources, based on a scalable spatial indexing and a decentralized security and authentication mechanism.

OGB allows anyone to publish ITS resources data relevant to a specific geographic area, ranging from transport schedules to sensor-generated or user generated real-time information, but also, point of interests, or planning services operated in the region. Users/travel operators/planning services/orchestrators can search and retrieve all resources available in an area of interest, and use it, for instance, to plan an optimal multimodal trip.

OGB exploits the Internames Communication System, an Information Centric Network (ICN) developed in WP3, to collect and make available geo-referenced transport-related resources in an efficient way.

OGB data model is centred on the GeoJSON standard. GeoJSON is a geospatial data interchange format based on JavaScript Object Notation (JSON). It defines several types of JSON objects and the manner in which they are combined to represent data about geographic Features, their properties, and their spatial extents. GeoJSON uses a geographic coordinate reference system, World Geodetic System 1984, and units of decimal degrees.

By representing travel data sources via GeoJSON Features, we have the possibility to describe the spatial extents and/or locations the travel data refers to. This information is used in OGB as the kernel indexing method, allowing scalability and fast search based on spatial properties. Additionally, other metadata have to be added to such description, by using more GeoJSON properties, in order to describe the data source in terms of meaningful keywords and, more importantly, by describing the networking end-point were the actual data are to be fetched from.

Thus, OGB easily accommodates both static data sources (such as GFTS or NeTEx feeds hosted at particular URLs) and dynamic ones, which we make available through the BONVOYAGE publish/subscribe mechanisms.

It is also able to index the availability of more generic services that are semantically related to an area. For instance, we use it to register the *soloists* (that is the set of independent travel planners that are invoked by the *orchestrator* in order to compute itineraries for each request coming into BONVOYAGE from its users, see D4.1), thus realizing a BONVOYAGE Directory Service (BVDS, see section 3.2.5).

All of the above is possible because the URL representing the network endpoint can be of a different nature according to the nature of the source itself: for instance, a name representing a static HTTP URL, or a name representing a dynamic Internames resource, or a TCP/IP endpoint to establish a communication channel with a given soloist.

This is accomplished by requiring two default properties each time a resource is inserted in OGB for indexing purposes:

- the **type** property indicates the nature of URL endpoint of the data being indexed. As of now valid values as FILE, CHANNEL, and SERVICE
- the **url** property indicates the network endpoint which must be used to fetch data published or made available by the transport operator, or to contact the soloist.

The FILE and CHANNEL types capture the dynamicity of the underlying information. A FILE data source can be expected to change rarely. Its data is thus published as a downloadable compressed feed, bundled as a single file, which is retrieved by a one-shot GET-like operation from the network. It is duty of the user to periodically re-scan the network URL associated with the data source to detect changes. Typically, a FILE data source covers a large geographic area and represents a massive amount of information. Conversely, a CHANNEL data source continuously changes in time. Its data is thus published as constant feeds under a specific "channel name", to which interested subscribers can attach and receive updates, that is they receive the latest fresh news that refer to that channel, as soon as the publisher pushes them and as incremental changes with respect to what was previously received.

The publish/subscribe primitives of the BONVOYAGE Internames Communication System are needed to exploit the CHANNEL capability, while the FILE capability is available through a classical HTTP GET scheme of operation. The SERVICE capability can be a TCP/IP socket or a RESTful endpoint, for instance.

Another default property is automatically added to the GeoJSON Feature that represents the resource, as soon as it is inserted into OGB. We call this property **OID** (Object IDentifier). So, OID is a unique ID automatically added by the OGB system when inserting, for instance:

OID = /OGB/014/038/20/61/GPS_id/GEOJSON/transitService/GTFS/admin/75185r2q57ye81t4

It uniquely identifies the GeoJSON Feature object that maps back to the real resource in the OpenGeoBase distributed system: it is an ICN name (internally used by the OGB backend, which is constructed using ICN technology, see D3.1) representing vital information about the geographic positioning of the elements of the resource and other needed information about the creator of the index and access rights (like tenant id, user id and collection id). It is a hierarchical name made-up of different parts (or components).

Specifically, the first part of OID name (e.g. /OGB/014/038/20/61/GPS_id in the example above) is created starting from latitude and longitude coordinates of the geometry elements (Point, MultiPoint, Polygon) we employ in the GeoJSON Feature. The special name component "GEOJSON" is just a fixed tag representing the inserted data format, it never changes. The next three components of the name are:

- Tenant id (e.g. "transitService")
- Collection id (e.g. "GTFS")
- User id (e.g. "admin")

The last component is a nonce added in order to make OID unique.

3.2 Metadata Handling Tool

The MDHT is the BONVOYAGE module that examines external resources, either by parsing (and possibly converting) feeds from sources of transport data or by talking to newly available services, and then extracts metadata useful for their indexing and provides such metadata to the OpenGeoBase, where they will be used by the different BONVOYAGE functions to discover and attach to the corresponding resource.

The following sections 3.2.1 and 3.2.3 show exemplary design of how MDHT maps static GTFS and NeTEx travel data sources into FILE-type OGB features. Section 3.2.5 shows how MDHT maps soloist services into SERVICE-type OGB features. The next two chapters show instead how the orchestrator (chapter 4) and the more dynamic travel-data sources (chapter 5) can exploit the discovery service for their own purposes.

3.2.1 Discovery of GTFS data

Until 2005, transit data lacked a common data format that could be used to share and integrate information among multiple agencies. In 2005, however, Google worked with Tri-Met in Oregon to create the General Transit Feed Specification (GTFS), an open data format now used by all transit agencies that participate in Google Maps[1]. GTFS feeds contain data for scheduled transit service including stop and route locations, schedules and fare information. The broad adoption of GTFS by transit agencies has made it a de facto standard. Those agencies using it are able to participate in a host of traveller services designed for GTFS, most especially transit trip planners.

GTFS defines a common format for public transportation schedules and associated geographic information. GTFS "feeds" allow public transit agencies to publish their transit data and developers to use that data to write applications. The feeds are represented in a series of text files that are compressed into a ZIP file, and include information such as fixed-route schedules, routes, and bus stop data. GTFS datasets are used in a variety of types of applications, including trip planners such as Google Maps, mobile applications, timetable generation software, tools for transit planning and operations analysis, and in many other categories of applications.

Availability of worldwide repositories of GTFS data has been intermittent during the first half of our project. Our main source during the initial phase has been the GTFS Data Exchange site (http://www.gtfs-data-exchange.com). It has shut down on September 2016.

We are now transitioning the MDHT to feed from other two main sources. TransitLand (https://transit.land) has steadily been growing as a reference spot for worldwide GTFS data, and the spot for transit agencies to connect with developers. **TransitFeeds** (http://transitfeeds.com) also has a comprehensive list of GTFS feeds, and is built on a much more open and transparent community contribution process via Github.

The MDHT approach is to monitor the above main transport information repositories, as well as other sources (coming, for instance, from project's partners such as Trenitalia), and to fetch GTFS data using the endpoints of the provided APIs.

Currently the BONVOYAGE "GTFS" CollectionID within OGB holds a distributed database of more than 7 million entries and indexes about 1000 GTFS files, but will become much bigger once the transition to the new feeds is completed. It

[1] Wong, James C., Watkins, Kari E.; December 2013; Use of the general transit feed specification (GTFS) in transit performance measurement; Georgia Institute of Technology.

allows fast searching at any spatial scale and primarily serves the purpose of providing baseline static public transport schedules to the rest of the components of the BONVOYAGE architecture.

3.2.2 GTFS mapping to GeoJSON

The MDHT operates on a GTFS feed, which is a collection of CSV files (with extension .txt) contained within a .zip file. Each file models a particular aspect of transit information: stops, routes, trips, and other schedule data. A feed contains at least 6 files, and up to 13 files: the names of these files and their structure are defined by the specification in order to obtain a valid GTFS feed. The GTFS specification defines the following files along with their associated content:

Filename	Required	Defines
agency.txt	Required	One or more transit agencies that provide the data in this feed.
stops.txt	Required	Individual locations where vehicles pick up or drop off passengers.
routes.txt	Required	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	Required	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
stop_times.txt	Required	Times that a vehicle arrives at and departs from individual stops for each trip.
calendar.txt	Required	Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.
calendar_dates.txt	Optional	Exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt.
fare_attributes.txt	Optional	Fare information for a transit organization's routes.
fare_rules.txt	Optional	Rules for applying fare information for a transit organization's routes.
shapes.txt	Optional	Rules for drawing lines on a map to represent a transit organization's routes.
frequencies.txt	Optional	Headway (time between trips) for routes with variable frequency of service.
transfers.txt	Optional	Rules for making connections at transfer points between routes.
feed_info.txt	Optional	Additional information about the feed itself, including publisher, version, and expiration information.

The following diagram shows the relationships among the various files:

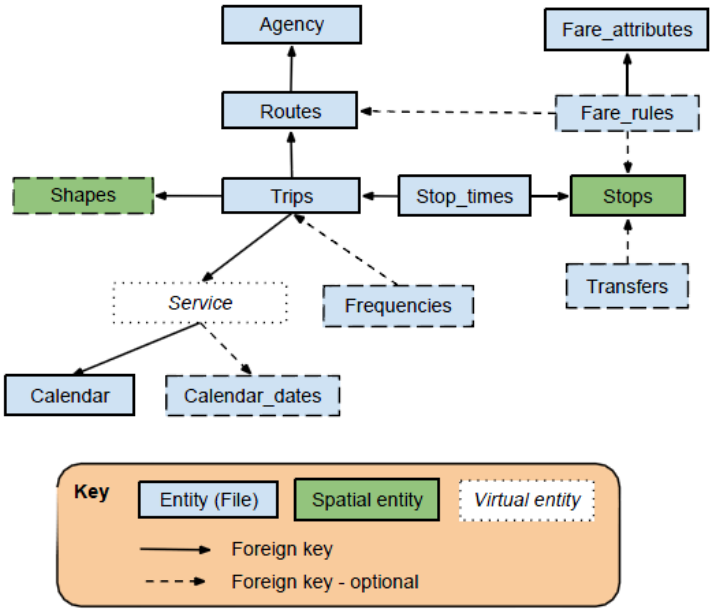

Figure 9: Structure of GTFS feeds

The following requirements apply to the format and contents of GTFS files:

- All files in a General Transit Feed Spec (GTFS) feed must be saved as comma-delimited text.
- The first line of each file must contain field names. Each reference page corresponds to one of the files in a transit feed and lists the field names you may use in that file.
- All field names are case-sensitive.
- Field values may not contain tabs, carriage returns or new lines.
- Field values that contain quotation marks or commas must be enclosed within quotation marks. In addition, each quotation mark in the field value must be preceded with a quotation mark. This is consistent with the manner in which Microsoft Excel outputs comma-delimited (CSV) files.
- The following example demonstrates how a field value would appear in a comma-delimited file:
  - **Original field value**: Contains "quotes", commas and text
  - **Field value in CSV file**: "Contains ""quotes"", commas and text"
- Field values must not contain HTML tags, comments or escape sequences.
- Remove any extra spaces between fields or field names. Many parsers consider the spaces to be part of the value, which may cause errors.
- Each line must end with a CRLF or LF line-break character.
- Files should be encoded in UTF-8 to support all Unicode characters. Files that include the Unicode byte-order mark (BOM) character are acceptable. Please see the Unicode FAQ for more information on the BOM character and UTF-8.
- Zip the files in the feed.

The following are just a few examples of the categories of applications that use GTFS data for various purposes.

- Trip planning and maps

There are a variety of applications that assist a transit customer in planning a trip from one location to another using public transportation. These provide step-by-step information on how to use various transportation options to reach a custom destination.

- Timetable creation

These applications create a printable list of the agency's schedule in a timetable format. They can also take the form of an HTML friendly or plain-text timetable.

- Data visualization

Various applications provide graphic visualizations of transit routes, stops, and schedule data. They can provide details such as the walkability, the quality of public transportation serving the area, and relate those factors to third criteria specific to the service (i.e. apartments available in the area).

- Accessibility

These include applications that assist transit riders with disabilities in using public transportation.

- Real-time transit information

These applications use GTFS data along with a real-time information source to provide estimated arrival information to transit riders. Newer formats, such as GTFS real-time and SIRI, can be added as an extension to a basic GTFS format so transit agencies can share real-time information.

In order to insert transit data to the OpenGeoBase distributed database, a GTFS feed must be mapped into a GeoJSON structure.

In a GTFS feed, geospatial information about transit service is basically stored in stops.txt file; the following table describes the fields in the stops.txt file (the optional fields have been omitted).

Field Name	Details
stop_id	Contains an ID that uniquely identifies a stop or station. Multiple routes may use the same stop. The stop_id is dataset unique.
stop_name	Contains the name of a stop or station. Please use a name that people will understand in the local and tourist vernacular.
stop_desc	Contains a description of a stop. Please provide useful, quality information. Do not simply duplicate the name of the stop.
stop_lat	Contains the latitude of a stop or station. The field value must be a valid WGS 84 latitude.
stop_lon	Contains the longitude of a stop or station. The field value must be a valid WGS 84 longitude value from -180 to 180.

The **{stop_lat, stop_lon}** couple defines the geographic location of the stop on the surface of the earth: this information can be stored in a **multipoint** GeoJSON as shown in next figure:

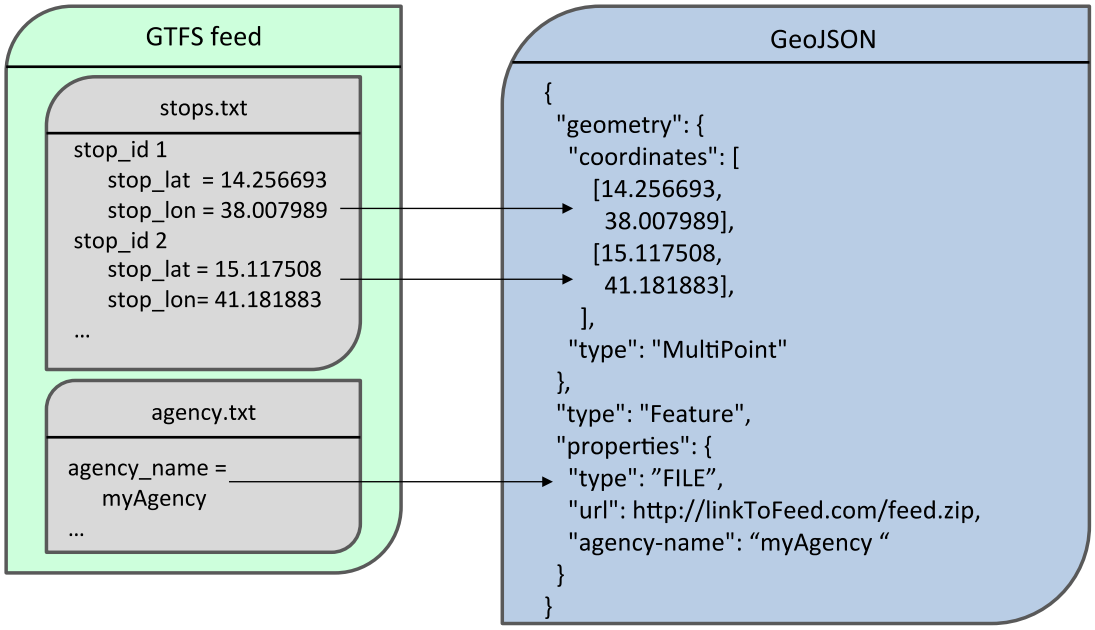

Figure 10: Mapping of a GTFS feed to GeoJSON

The *coordinates* field of a multipoint GeoJSON is an array of GPS points, and each one of them represents a stop defined in the file stops.txt of a GTFS feed.

In the *properties* field additional information about the transit agency and GTFS feed have to be stored.

For GTFS it is crucial to store the URL where the feed itself can be fetched, and to classify the source type as FILE. **The *url* field contains the URL of the transit agency**. The value must be a fully qualified URL that includes http:// or https://, and any special characters in the URL must be correctly escaped. As explained above, **the *type* property is set to FILE is this case**.

Additionally, the bare minimum information coming from the associated agency.txt feed is to be included. Hence, we decided to create a GTFS-specific property: *agency-name.* The *agency-name* field contains the full name of the transit agency, which is the same information stored in the agency-name field of the agency.txt file.

When all the parts of the GeoJSON object that corresponds to the GTFS record are constructed, the MDHT invokes the proper OGB API in order to add the object into the discovery database. In this case:

OGB.addMultiPoint(authentication_token, coordinates, properties, “GTFS”)

Is invoked, so that a MultiPoint geometry is created using the **coordinates** portion of the object and the relevant **properties** are associated with it. Please notice that a “GTFS” string is part of the API signature and invocation. It represents the OGB’s **CollectionID** where to store the object. The **authentication_token** contains relevant user and access control info.

3.2.3 Discovery of NeTEx data

NeTEx is an evolution of the Transmodel project, which developed a conceptual model to harmonise and systemise the data formats of a number of European countries. It is a very new standard and at the moment of this writing (February 2017) only a small number of transport operators have started drafting their data into NeTEx.

Within the BONVOYAGE project, partner NPRA is starting to provide NeTEx-formatted data about public transport schedules in Norway, but we are still at an early stage and data sources are not fully reliable, yet. Nonetheless we have started adapting the MDHT for NeTEx, because it is intended to be the reference European standard for static and semi-static public transport schedules publishing.

NeTEx provides a modular XML schema for public transport information data including passenger information systems, with coverage of a number of different subdomains of PT information, including transport network infrastructure and topology, public transport schedules, journey planning, fares, fare validation.

NeTEx is a CEN Technical Standard. It is free to use under a GPL License and the CEN standards process manages its development.

The NeTEx and GTFS formats should be considered as complementary, covering different stages in the data management process. NeTEx has a much wider scope, and it is intended for use in back office use cases under which data is generated, refined and integrated with external services.

Because it uses XML, NeTEx is able to package a complete data set as a single coherent document that can be managed and validated. GTFS uses a traditional flat file format: this is compact and efficient but requires multiple files to describe the different types of element and thus additional rules for naming and packaging the files as a zip, as explained above.

It is possible to generate a full GTFS data set from NeTEx but not vice versa. The NeTEx UML includes a GTFS mapping package, which shows how each GTFS element may be populated from the corresponding NeTEx element.

All NeTEx information is available from http://netex-cen.eu/ where all the schemas and papers explaining the NeTEx XML design can be consulted. It is possible to get basic synthetic examples, too.

To better understand how the MDHT comparatively acts on NeTEx and GTFS when feeding metadata to OGB, we examine in the following paragraph two NeTEx examples (about agency and stops), which have a direct mapping from NeTEx to GTFS, hence generating homogeneous metadata for OGB.

The example XML (NeTEx) and .txt files (GTFS) are available within a .zip bundle from the official NeTEx website, and can be download from http://netex-cen.eu/wp-content/uploads/2016/02/NeTEx-XML-master-1-03.zip. The relevant files are:

- NeTEx-XML-master-1-03.zip\NeTEx-XML-master\schema\1.03\examples\standards\gtfs\Netex_gtfs_exm1_Agency_1.xml
- NeTEx-XML-master-1-03.zip\NeTEx-XML-master\schema\1.03\examples\standards\gtfs\Netex_gtfs_exm1_Stops_1.xml

Hereby we only examine the relevant portions. The Netex_gtfs_exm1_Agency_1.xml example provides Agency information. NeTEx XML follows:

```
<PublicationDelivery version="1.0" xsi:schemaLocation="http://www.netex.org.uk/netex ../../../xsd/NeTEx_publication.xsd"
xmlns="http://www.netex.org.uk/netex" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
        <PublicationTimestamp>2001-12-17T09:30:47.0Z</PublicationTimestamp>
        <ParticipantRef>SYS001</ParticipantRef>
        <!--- ======WHAT WAS REQUESTED ========== -->
        <PublicationRefreshInterval>P3M</PublicationRefreshInterval>
        <Description>Example of GTFS Agency  data</Description>
        <!--- ============== RESULTS =========== -->
        <dataObjects>
                <ResourceFrame version="any" id="mygtfsxm:ResourceFrame:DTA">
                        <!--- ======= CODESPACEs======== -->
                        <codespaces>
                                <Codespace id="mygtfsxm">
                                        <Xmlns>mygtfsxm</Xmlns>
                                        <XmlnsUrl>http://www.demoagency.com/</XmlnsUrl>
                                        <Description>Demo Agency</Description>
                                </Codespace>
                        </codespaces>
                        <!--- =======FRAME DEFAULTS======== -->
                        <FrameDefaults>
                                <DefaultCodespaceRef ref="mygtfsxm"/>
                        </FrameDefaults>
                        <organisations>
                                <!--- =====Agency========= -->
                                <Authority version="any"  id="DTA">
                                        <!-- Agency.txt.agency_id  -->
                                        <!-- Agency.txt.agency_name -->
                                        <Name>Demo Transit Authority</Name>
                                        <Locale>
                                                <!-- Agency.txt.time_zone -->
                                                <TimeZone>America/Los_Angeles</TimeZone>
                                                <DefaultLanguage>en</DefaultLanguage>
                                        </Locale>
                                        <ContactDetails>
                                                <Phone>1-212-675-9876</Phone>
                                                <!-- Agency.txt.agency_url -->
                                                <Url>http://google.com</Url>
                                        </ContactDetails>
                                </Authority>
                        </organisations>
                </ResourceFrame>
        </dataObjects>
</PublicationDelivery>
```

The Netex_gtfs_exm1_ Stops _1.xml example provides data about Stops. NeTEx XML follows, cut to only show the section referencing two points relevant for our OGB mapping below):

```
<ScheduledStopPoint version="any" id="mygtfsxm:ScheduledStopPoint:FUR_CREEK_RES">
```

```
        <Name>Furnace Creek Resort (Demo)</Name>
        <Location>
                <Longitude>-117.133162</Longitude>
                <Latitude>36.425288</Latitude>
        </Location>
        <types>
                <TypeOfPointRef version="any" ref="gtfs:TypeOfPoint:0"/>
        </types>
        <tariffZones>
                <TariffZoneRef version="any" ref="mygtfsxm:TariffZone:FareZone01"/>
        </tariffZones>
        <PublicCode>121</PublicCode>
                <Url>http://www.demoagency.org</Url>
                <StopType>onstreetBus</StopType>
</ScheduledStopPoint>
<ScheduledStopPoint version="any" id="mygtfsxm:ScheduledStopPoint:BEATTY_AIRPORT">
        <Name>Nye County Airport (Demo)</Name>
        <Location>
                <Longitude>-116.784582</Longitude>
                <Latitude>36.868446</Latitude>
        </Location>
        <types>
                <TypeOfPointRef version="any" ref="gtfs:TypeOfPoint:0"/>
        </types>
        <tariffZones>
                <TariffZoneRef version="any" ref="mygtfsxm:TariffZone:FareZone01"/>
        </tariffZones>
        <!-- Stops.txt  -(google stop_code (google valuue) -->
        <PublicCode>122</PublicCode>
        <Url>http://www.demoagency.org</Url>
        <StopType>onstreetBus</StopType>
</ScheduledStopPoint>
```

3.2.4 NeTEx mapping to GeoJSON

We can represent the NeTEx mapping to the GeoJSON data structure we need to construct for OGB, as follows:

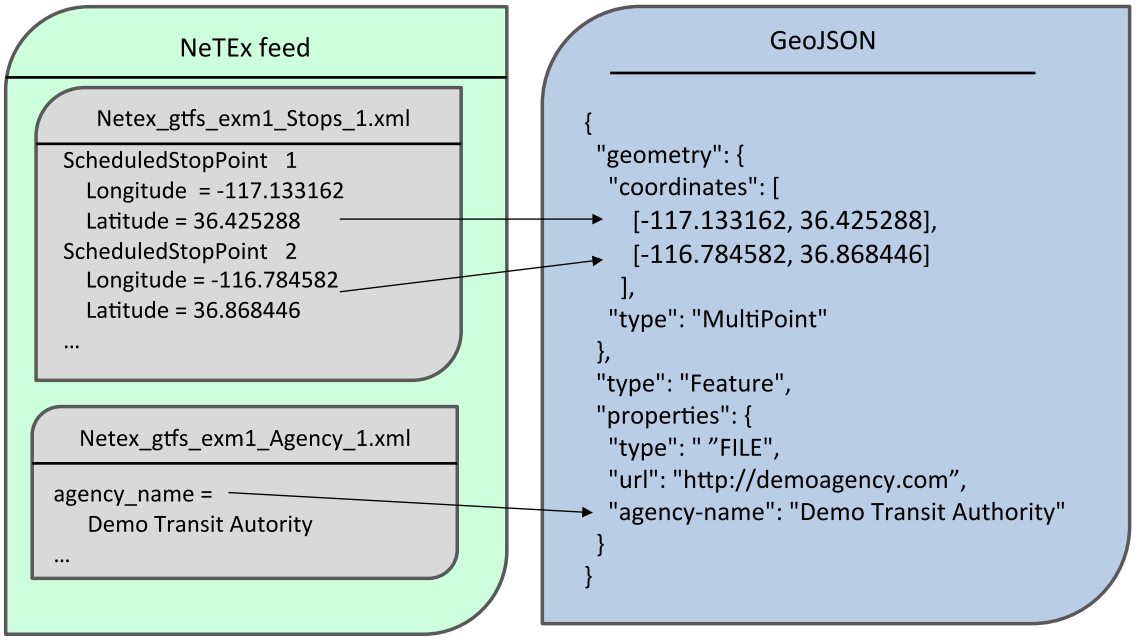

Figure 11: Mapping of a NeTEx feed to GeoJSON

As it was the case for GTFS, the *coordinates* field of our multipoint GeoJSON is an array of GPS points, each one of them representing a stop defined in the NeTEx XML feed that represents the Stops.

In the *properties* field it is crucial to store the URL where the NeTEx feed itself can be fetched, and to classify the source type as FILE. **The *url* field contains the URL of the transit agency**. The value must be a fully qualified URL. As explained above, **the *type* property is set to FILE is this case**.

Additionally, the bare minimum information coming from the XML NeTEx feed representing the Agency is to be included. For clarity we just include the name here, under the *agency-name* property. The NeTEx .XSD schemas we are using so far are available at: NeTExXml-v1.03\xsd\NeTEx_publication.xsd.

When all the parts of the GeoJSON object that corresponds to the NeTEx record are constructed, the MDHT invokes the proper OGB API in order to add the object into the discovery database. In this case:

OGB.addMultiPoint(authentication_token, coordinates, properties, "NETEX")

Is invoked, so that a MultiPoint geometry is created using the **coordinates** portion of the object and the relevant **properties** are associated with it. Please notice that a "NETEX" string is part of the API signature and invocation. It represents the OGB's **CollecionID** where to store the object. The **authentication_token** contains relevant user and access control info.

In conclusion, both in this case and the former we use a MultiPoint geometry, instantiating a bi-dimensional coordinate for each stop, **to capture the information about all bus stops that are included in the whole static feed file as a bulk**. Whenever OGB is asked about information available to its GTFS or NETEX collections, within a specified **area that includes one or more stops, the GeoJSON objects representing the stops are given back.**

Specifically, the following OGB API is invoked:

OGB. rangeQuery(authentication_token, CollectionID, SouthWest_Lat, SouthWest_Lon, area_size_degrees)

where the interest region is expressed by its south-west corner coordinates and its size in degrees. The wanted collection name must be specified, as well as authentication info.

The above API call gives back a list of matching GeoJSON objects. The user can then proceed to autonomously download the file from the given URL of the **corresponding feed, included in the GeoJSON URL property**.

3.2.5 Discovery of trip planning soloists and BONVOYAGE Directory Service

The orchestrator service distributes trip-planning tasks to multiple soloist, i.e. local planning services. The orchestrator must therefore know, at any point in time, which services are running at that time, and for each service, the type of service, the URL, and communication interface format. Hence, we introduced a BONVOYAGE trip planning Directory Service (BVDS), fully based on the discovery procedures we illustrated above for static GTFS and NeTEx data sources. In this way, an orchestrator can dynamically maintain its overlay graph (that connects all relevant soloists, as we will explain in section 4.2.1) up-to-date.

The MDHT is in charge of talking to every soloist wanting to be listed in BVDS, in order to gather a core set of metadata regarding the soloist, sufficient for the orchestrator to find it and trigger a communication channel with it.

Specifically the following minimum set of information has to be gathered:

- boundaries of the geographical area the soloist is able to cover with its algorithm.
- URL to reach out to the soloist APIs.
- Format of the messages to be exchanged over the communication channel.

These characteristics of the soloist are then mapped by the MDHT into a GeoJSON that represents the soloist as a resource in OGB, allowing the orchestrator to express interest in a geographic region, and to discover what services are available in there, as well as their contact network endpoints.

Different than the GTFS and NeTEx mappings we illustrated above, for soloists the ***type*** property of the GeoJSON Feature is going to obviously be a SERVICE type, and the ***geometry type*** is going to be a **Polygon instead** of a Point or MultiPoint.

In addition to the above minimum set, optional properties can be specified in order to distinguish different categories of SERVICEs. For instance, when the SERVICE to be indexed is not a soloist (it can be an orchestrator for instance, since our design does accommodate, in principle, for multiple orchestrators to be exploited by the BONVOYAGE platform), then the optional ***service-type*** property can be assigned the “orchestrator” value.

We have followed the definition of Polygon GeoJSON geometries, as it is defined in the standard. Specifically, to specify a constraint specific to Polygons, it is useful to introduce the concept of a linear ring:

- A linear ring is a closed line with four or more positions.
- The first and last positions are equivalent, and they must contain identical values.
- A linear ring is the boundary of a surface or the boundary of a hole in a surface.

- A linear ring must follow the right-hand rule with respect to the area it bounds, i.e., exterior rings are counter clockwise, and holes are clockwise.

The specification does not discuss linear ring winding order, but it is recommended that for backwards compatibility, parsers should not reject Polygons that do not follow the right-hand rule.

Though a linear ring is not explicitly represented as a GeoJSON geometry type, it leads to a canonical formulation of the Polygon geometry type definition as follows:

- For type "Polygon", the "coordinates" member must be an array of linear ring coordinate arrays.
- For Polygons with more than one of these rings, the first must be the exterior ring, and any others must be interior rings. The exterior ring bounds the surface, and the interior rings (if present) bound holes within the surface.

The following section presents an example mapping of a soloist service to a GeoJSON OGB Feature.

3.2.6 Soloist mapping to GeoJSON and wrapper API

An exemplary soloist, which is able to operate on the Rome region at large and has been developed by BONVOYAGE partner CRAT would be mapped to a GeoJSON Feature as follows:

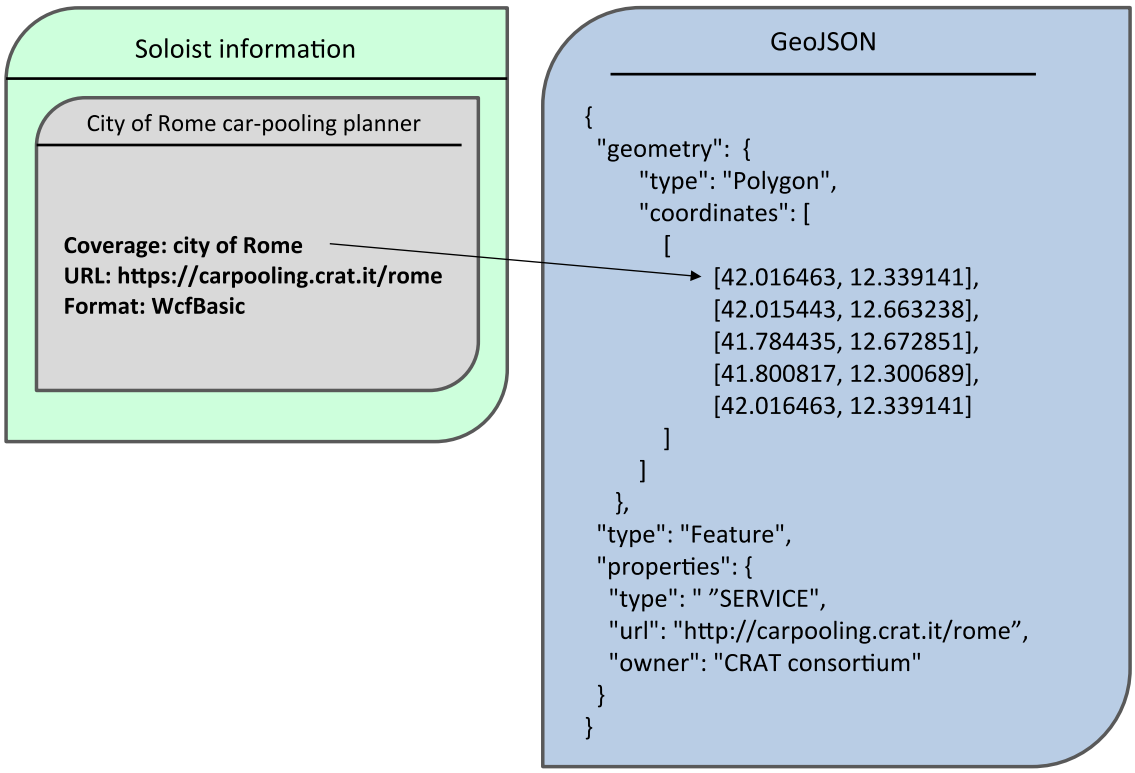

Figure 12: Mapping of a Soloist to GeoJSON

Please notice that the last coordinate coincides with the first one, as the standard requires.

When all the parts of the GeoJSON object that corresponds to the soloist's boundaries are constructed, the MDHT invokes the proper OGB API in order to add the object into the discovery database. In this case:

OGB.addPolygon(authentication_token, coordinates, properties, "SERVICES")

Is invoked, so that Polygon geometry is created using the **coordinates** portion of the object and the relevant **properties** are associated with it. Please notice that a "SERVICES" string is part of the API signature and invocation. It represents the OGB's **CollecionID** where to store the object, and we decided to group all the resources that represent services (i.e. both Soloists and Orchestrators) under the "SERVICES" collection. The **authentication_token** contains relevant user and access control info.

On top of the above mapping, a simplified wrapper API, which we call the BONVOYAGE Directory Service, was developed in order for the platform to conveniently query the OGB backend about the availability of all services in a rectangular region:

Protocol	http POST
Method	`GetServices(param)`

Parameters	`"param" : {`
	`  "boundingBox" : [`
	`  "topLeftLat" : "number",`
	`  "topLeftLon" : "number",`
	`  "bottomRightLat" : "number",`
	`  "bottomRightLon" : "number"`
	`  ],`
	`  "returnAll" : "boolean"`
	`}`

Response	`{[`
	`  {`
	`  "urls" : [`
	`  "url" : "string",`
	`  "format" : { "enum" [ "Xml", "Json", "WcfBasic" ] }`
	`  ],`
	`  "serviceType" : "enum" [ "Orchestrator", "Soloist" ]`
	`  }`
	`]}`

The GetServices API of the BVDS directly maps onto an equivalent call of the OGB.rangeQuery() API.

A typical answer from the BVDS is shown if the figure below. Please notice that the current setup includes four soloists and a reference orchestrator. All of them are registered as different services within the BVDS.

Services in selected directory

Type	Url	Format
Soloist	http://bonvoyage.sintef.no/Soloist/DemoGoogle/basic	WcfBasic
	http://bonvoyage.sintef.no/Soloist/DemoGoogle/json	Json
	http://bonvoyage.sintef.no/Soloist/DemoGoogle/xml	Xml
Orchestrator	http://bonvoyage.sintef.no/Routing/basic	WcfBasic
	http://bonvoyage.sintef.no/Routing/json	Json
	http://bonvoyage.sintef.no/Routing/xml	Xml
Soloist	http://bonvoyage.sintef.no/Soloist/DemoBilbao/basic	WcfBasic
	http://bonvoyage.sintef.no/Soloist/DemoBilbao/json	Json
	http://bonvoyage.sintef.no/Soloist/DemoBilbao/xml	Xml
Soloist	http://bonvoyage.sintef.no/Soloist/DemoAirplane/basic	WcfBasic
	http://bonvoyage.sintef.no/Soloist/DemoAirplane/json	Json
	http://bonvoyage.sintef.no/Soloist/DemoAirplane/xml	Xml
Soloist	http://bonvoyage.sintef.no/Soloist/DemoGrenoble/basic	WcfBasic
	http://bonvoyage.sintef.no/Soloist/DemoGrenoble/json	Json
	http://bonvoyage.sintef.no/Soloist/DemoGrenoble/xml	Xml

Figure 13: A formatted response from the BVDS GetServices API

4 Adaptation of trip planning services

This section explains how the trip planning solvers designed as part of Work Package 4 are adapted into services. The algorithmic design of the trip planning task is a hierarchical, decomposition-based approach where one or more *orchestrators* handles the door-to-door trip requests, decomposes each trip request into sub-requests that are forwarded to soloists, collects the corresponding sub-responses, assembles the routes of the sub-responses into complete routes, ranks the routes and finally returns the response of the trip request. The *soloists* are distributed to handle trip planning tasks for a part of the overall transportation and road network, for a sub-set of modalities or specialized for a single modality. In this way both trip planning soloists developed within the BONVOYAGE project as well as soloists based on existing third party trip planning solvers can collaborate in the overall trip planning framework.

Figure 14 shows the layered view of the BONVOYAGE platform used by the BONVOYAGE application. From the user perspective, a route query is set and personalised, and ranked solutions are returned through the Application Server, but the internal communication is supported by the SPROUTE format.

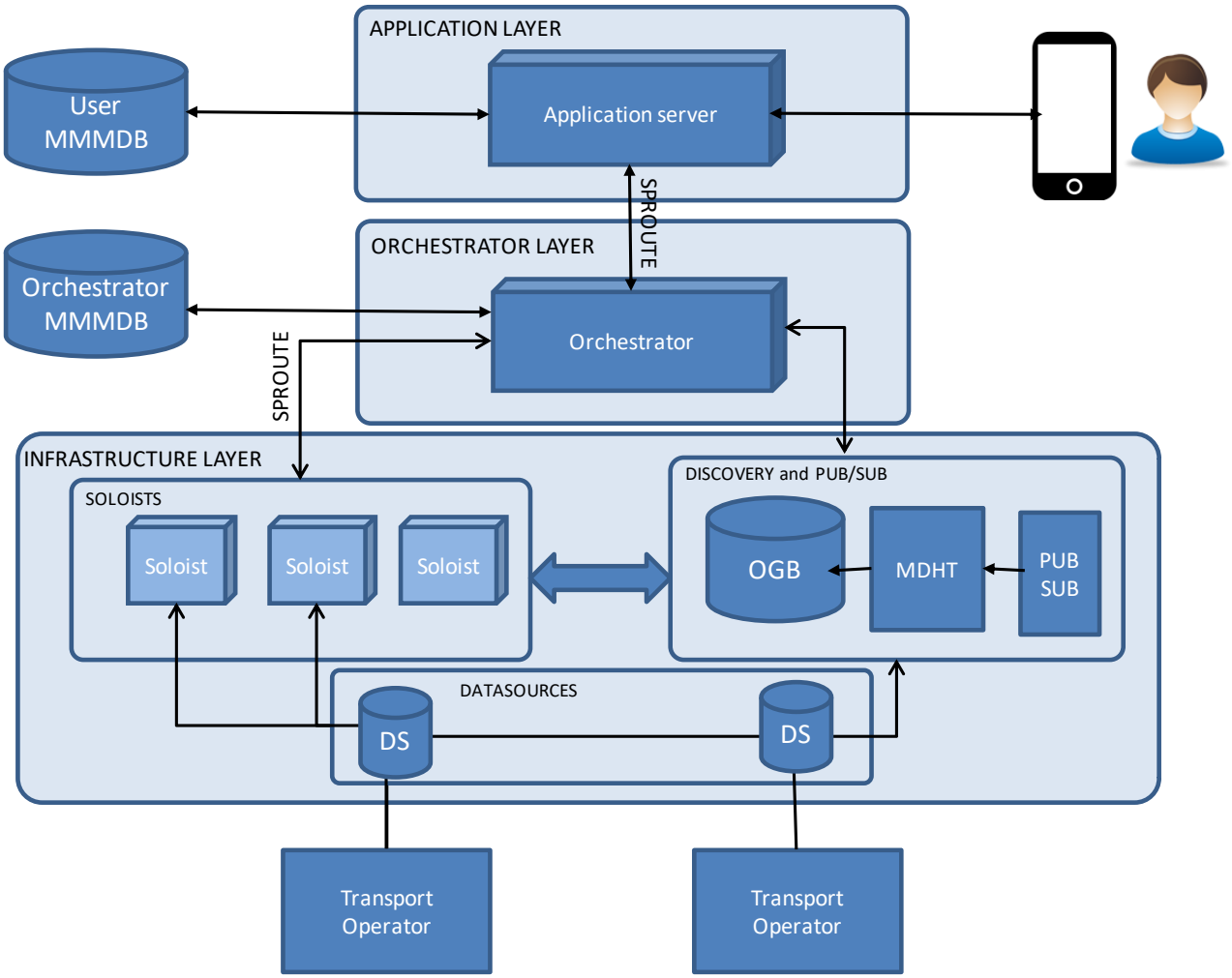

Figure 14: Layered view of BONVOYAGE

The first part of the work for adaptation of soloists and orchestrator as inter-linked services was the adaptation of the SPROUTE protocol, which transports routing information, to support all new optimisation and personalization features of the platform. Key requirements for that were identified in D2.1 *Use Cases and Reference Architecture* and further refined in D2.2 *BONVOYAGE Architecture*. Additional requirements were set by the work on Intelligent Transport Functionalities in WP4 with respect to the multi-object optimisation for route planning, the personalisation by user profiles and green score policy. To address these requirement adaptions of the routing interface were required.

4.1 Adaptation of the SPROUTE data format

As said, the starting point for the routing format was the SPROUTE format[2]. It is an open source format to exchange routing information (request and response) as JSON objects. It defines the geometry of trips but also detailed specification of the used mode of transports. Its openness and its application in other projects were reasons to select the format as a base line for the BONVOYAGE platform. Adaptions were required to fully support the functionalities provided by BONVOYAGE.

The SPROUTE format is defined as JSON schema defining the structure and elements. The format consists of two main parts, the request and response. The fundamental concepts of the SPROUTE format are shown in Figure 23. The minimum required properties of a SPROUTE document to be valid are the coordination reference system, a required ID, the format version, the status of the document, a timestamp and a route element. Figure 23 lists all concepts defined the concepts defined in the SPROUTE data format.

The following concepts were added or modified of the original SPROUTE format to support the requirements of BONVOAGE and the personalisation of the route planning service.

- Travelling Entities
- Real-time routing
- Route leg constraints
- Costs of segments
- Trip reference
- Greenpoints

The travelling entities, the real-time routing, route leg constrain extend the concept of a route request. Costs of segments, trip reference and greenpoints are new concepts of the route response.

4.1.1 Travelling Entity

The RoutingRequest was extended by the optional property of a travel entity. Figure 15 shows the data structure of the travelling entity object. It enables the specification of travellers for a routing request. The travellers can be either persons or parcels. The format supports to define groups of travelling entities for a routing request (e.g. party or travelling with luggage). Each travel entity can be defined in detail (e.g. memberships, tickets). The tickets and memberships that a traveller holds can be used to optimise the travel costs for the individual conditions. The detail information of the travellers can be further detailed if required (e.g. age of travellers) by the data object additionalInfo.

Figure 16 shows the data object ticket. The tickets that are already held by the travellers can be defined in the routing request. It allows optimising the travel costs of the travellers. The ticket consists of an id, name, fare zone and valid period as shown in Figure 16. The ticket concept is also used to define re-routing routing request for on-going tips in case of interruptive events (e.g. traffic jam or delays) as described in Section 4.1.2.

[2] https://github.com/dts-ait/ariadne-json-route-format

Figure 15: SPROUTE data object: Traveling Entities

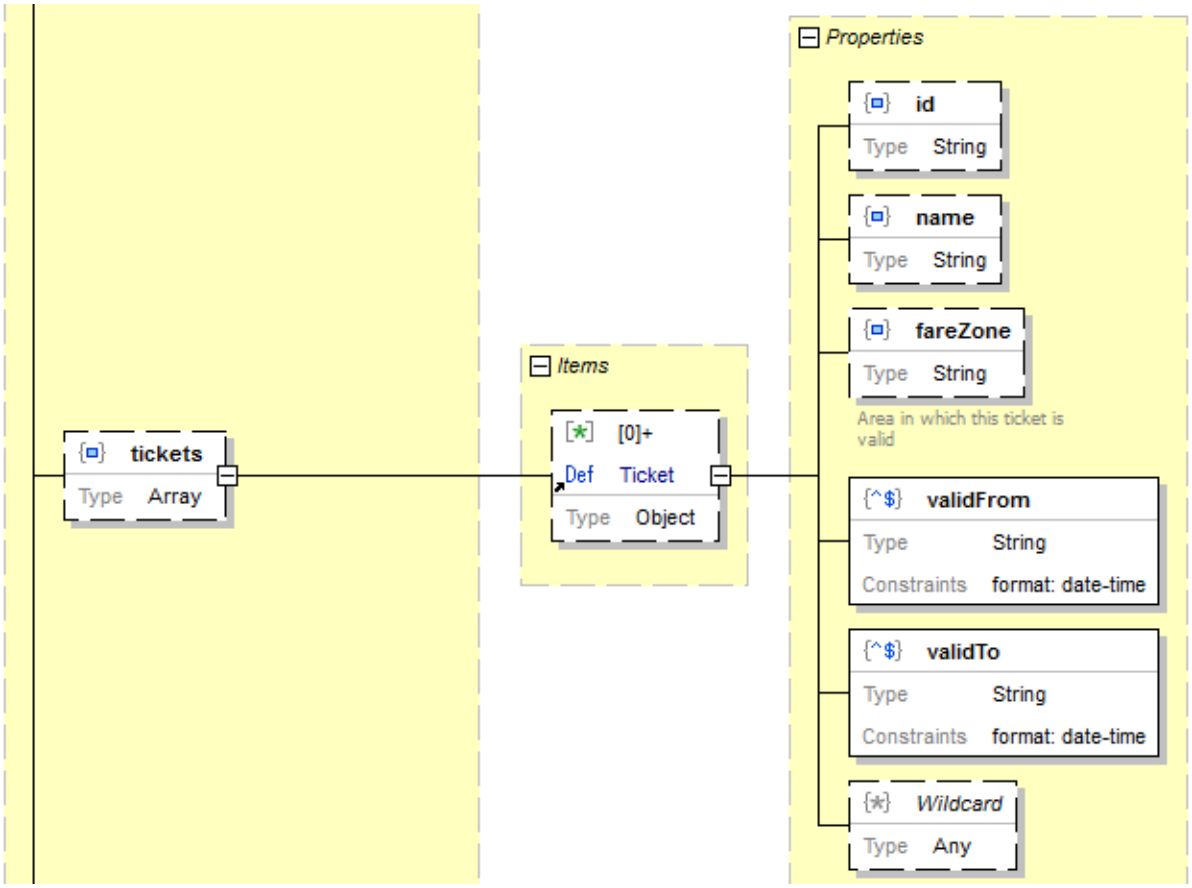

Figure 16: SPROUTE Data object: tickets

4.1.2 Real-time routing

To support real time routing the current route of the traveller is required. It allows to response to current events by planning of new alternative routes, e.g. avoiding traffic jams, delay of trains. For new alternative trip planning the trip so far needs to be considered, e.g. tickets bought for the current trip, modalities used, and time of travel. For example, travellers using their own car will not switch to public transport to reach their destination in most of the cases. The RoutingRequest was extended by the data object currentRoute including a Route object of the current route of the traveller, shown in Figure 17.

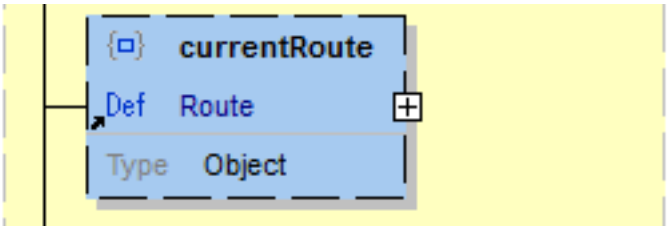

Figure 17: SPROUTE data object: currentRoute

4.1.3 Route leg constraints

The SPROUTE format was extended by constrains for route legs. The SPROUTE format allows to defined via-Points for the route requests. For BONVOYAGE we extended the constraints for the via-location by timeConstraint and segmentConstraint as shown in Figure 18. Figure 19 outlines the two new objects. The data object timeConstraint defines time constrains for each leg of the trip in terms of earliest and latest time. The segmentConstraint specifies constrains in terms of mode of transport. It defines mode of transport to use as well as to avoid for the segment. The data object segmentConstraint further allows the other constraint object such as price, duration, pollution, energy and distance. The constraints are defined for the “via” point and refer to the leg arriving at the via-point.

The objective concept as shown in Figure 19 is used to personalise the routing request by the user profiling tool. Based on the individual profile of the traveller the data object objective is used to enrich the routing request with optimisation objectives (e.g. costs for a student, time for business trip).

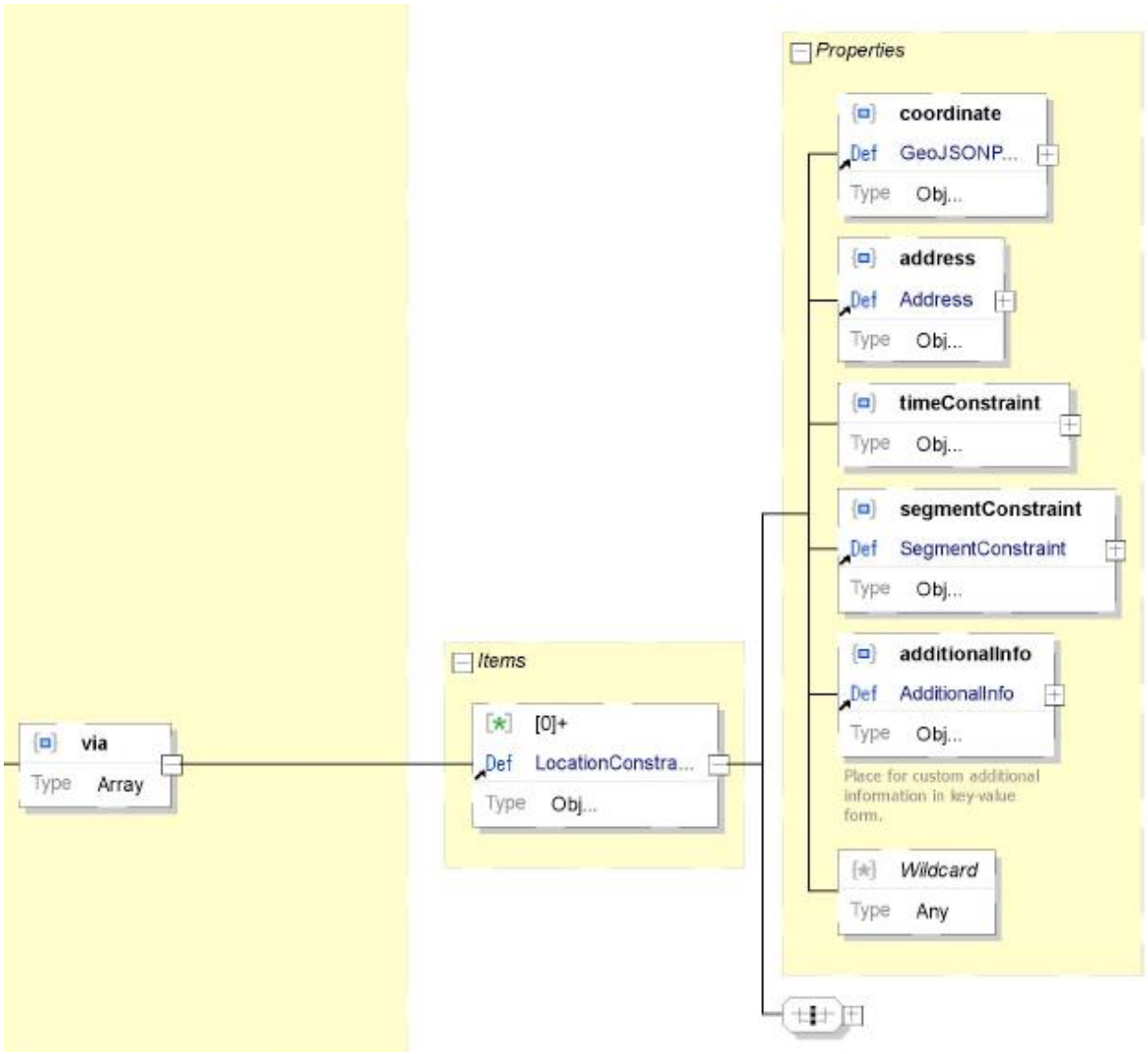

Figure 18: SPROUTE Data objects of RouteLeg Constrains

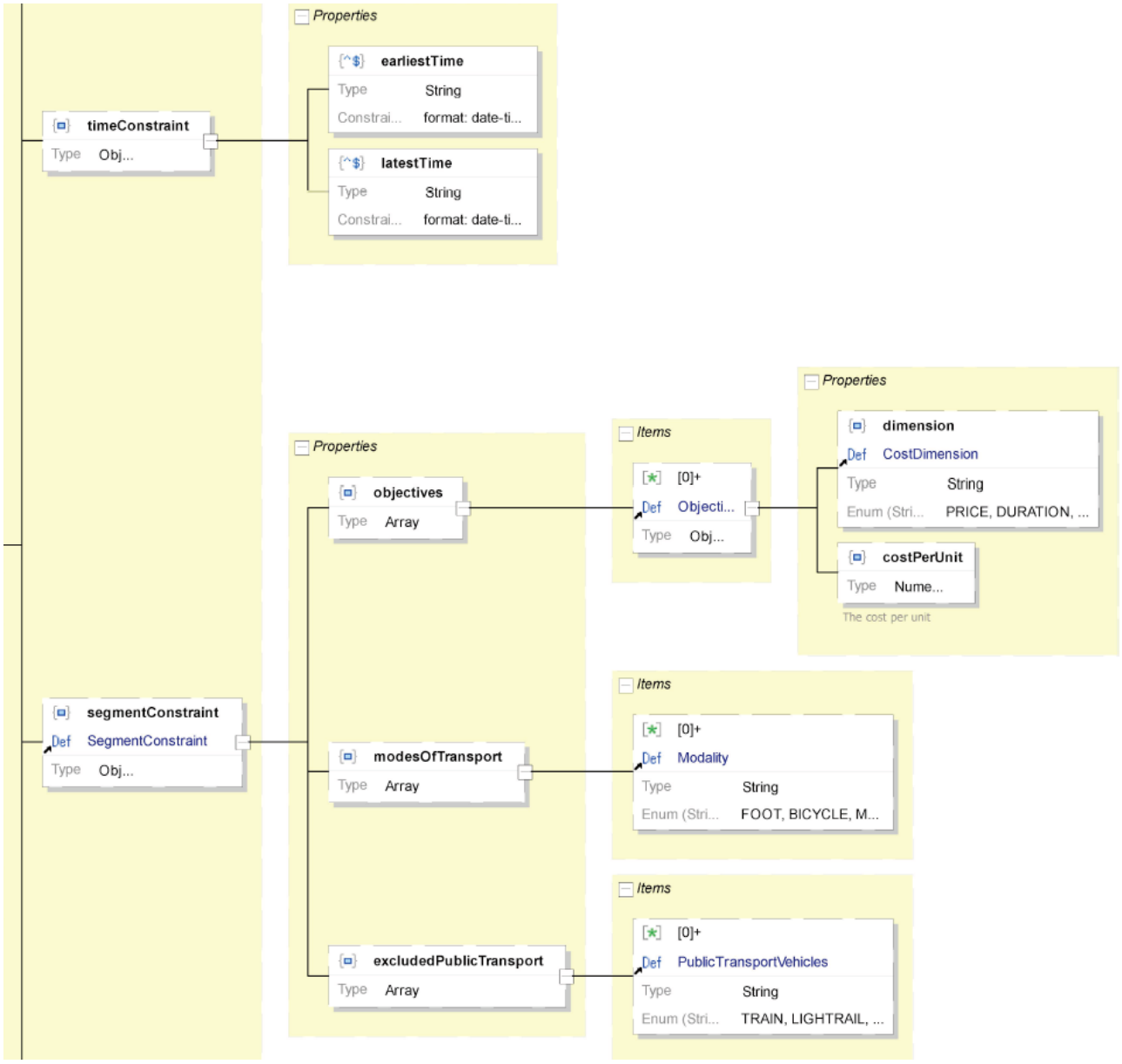

Figure 19: SPROUTE Data object segmentConstraint and timeConstraint

4.1.4 Costs of segments

The route object of the SPROUTE format was extended by the concepts of costs. The costs can be defined for the route and for each a route segment. Figure 20 shows the cost object, which allows specifying the weight of the cost (i.e. the unit cost) in different cost dimensions. The dimensions are price, duration, pollution, energy and distance.

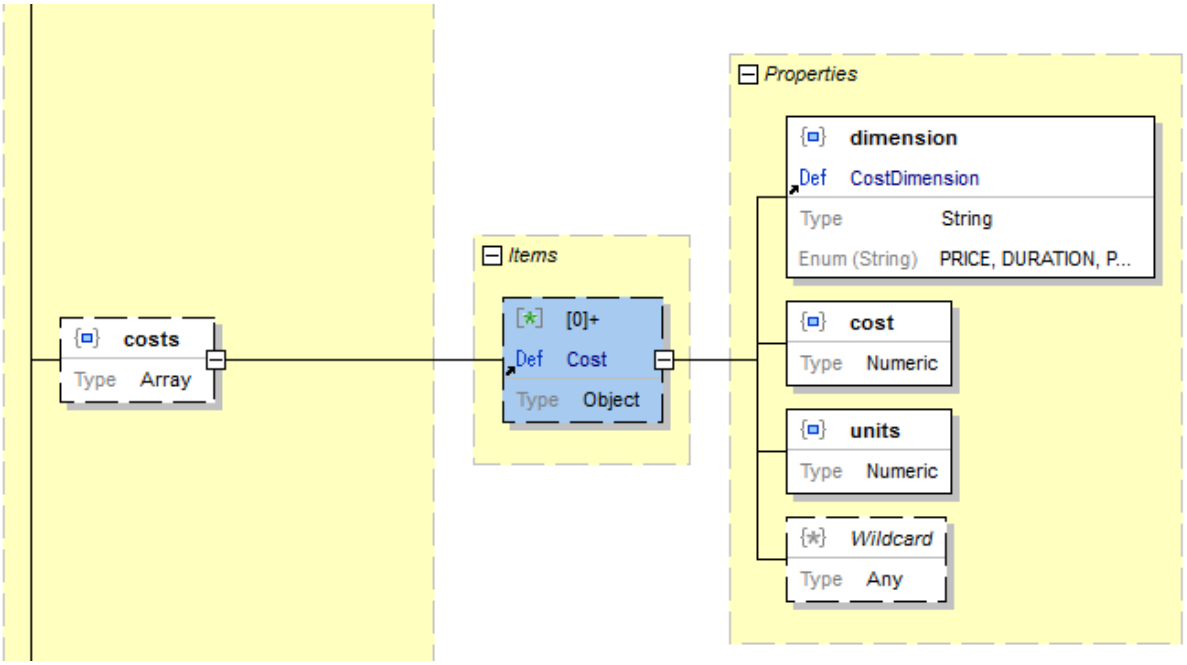

Figure 20: SPROUTE Data object costs

4.1.5 Trip reference

The mode of transport concepts was extended by more specific trip reference for trips that are provided as a service (e.g. public transport). A line id and a trip id were added to the service data object as shown in Figure 21. This concept allows to link to more specific service of the operator that can be used to provide more information to the individual service or monitor the real-time data of the service.

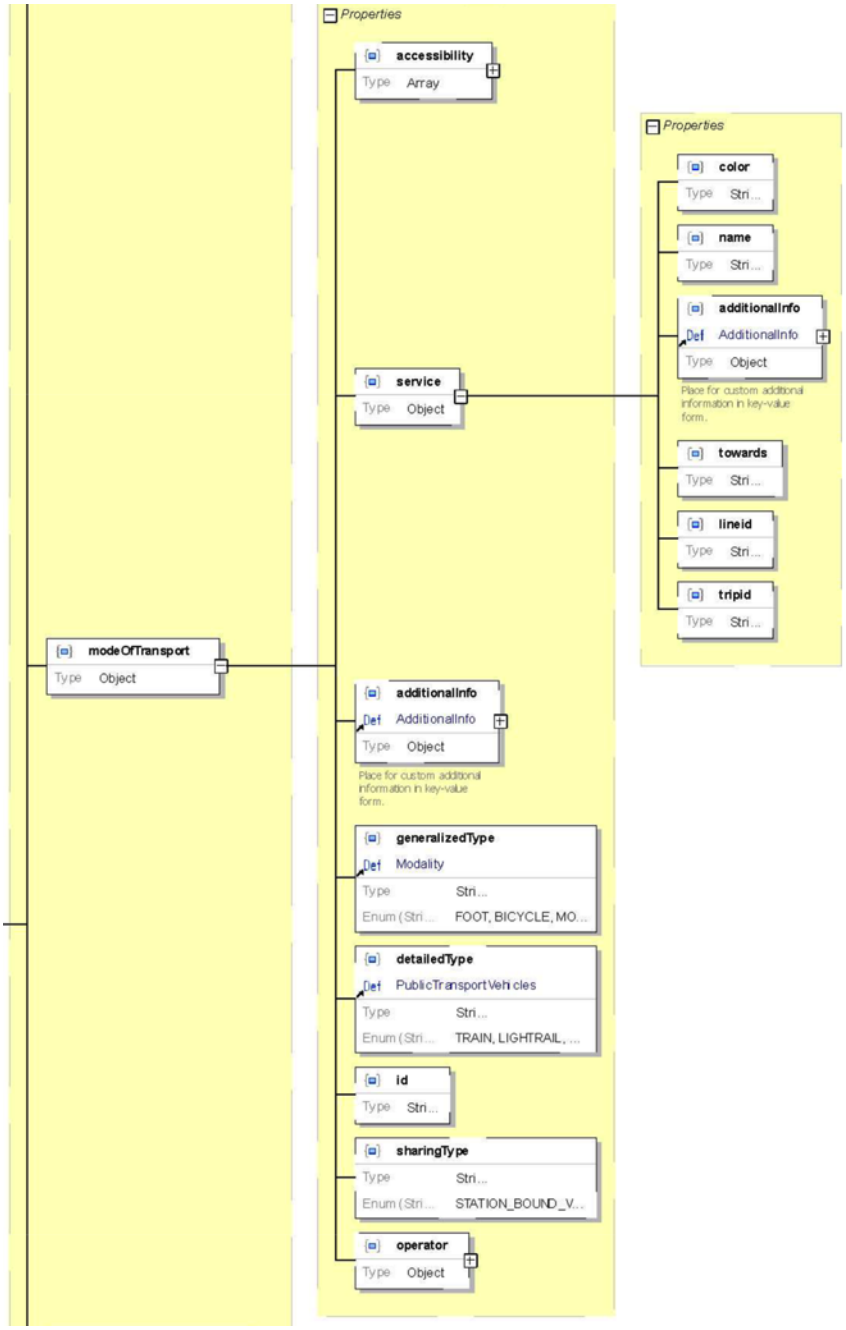

Figure 21: SPROUTE Data object service

4.1.6 Greenpoints

The BONVOYAGE score policy was introduced in D4.1 Design of the Intelligent Transport Functionality. Greenpoints and CO_2 values are calculated the route based on a user specific profile. The greenpoint values are added to SPROUTE definition for route and route segment as shown in Figure 22. The greenpoints of a route is a personalised calculation

based on the user profile. The greenpoints are used in the BONVOYAGE app to highlight the ecological impact of a route and provide a motivation for the user to select ecological sustainable travel choices.

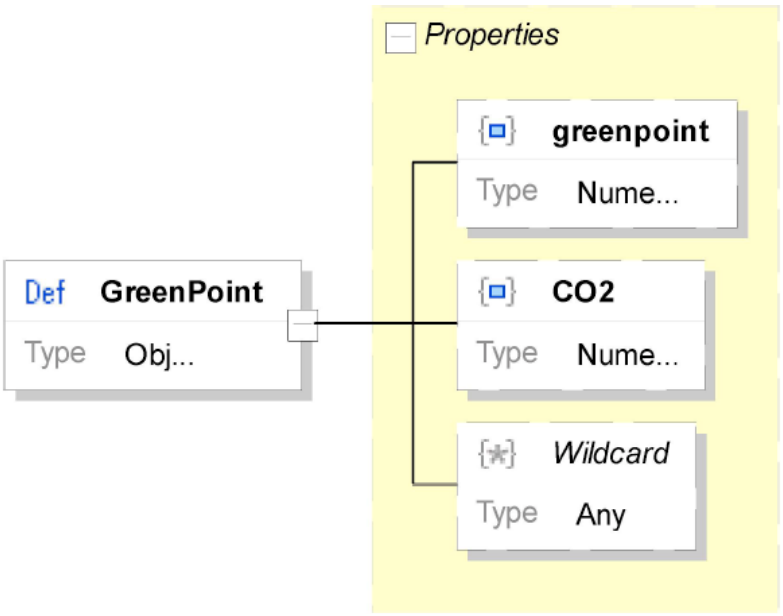

Figure 22: SPROUTE Data object GreenPoints

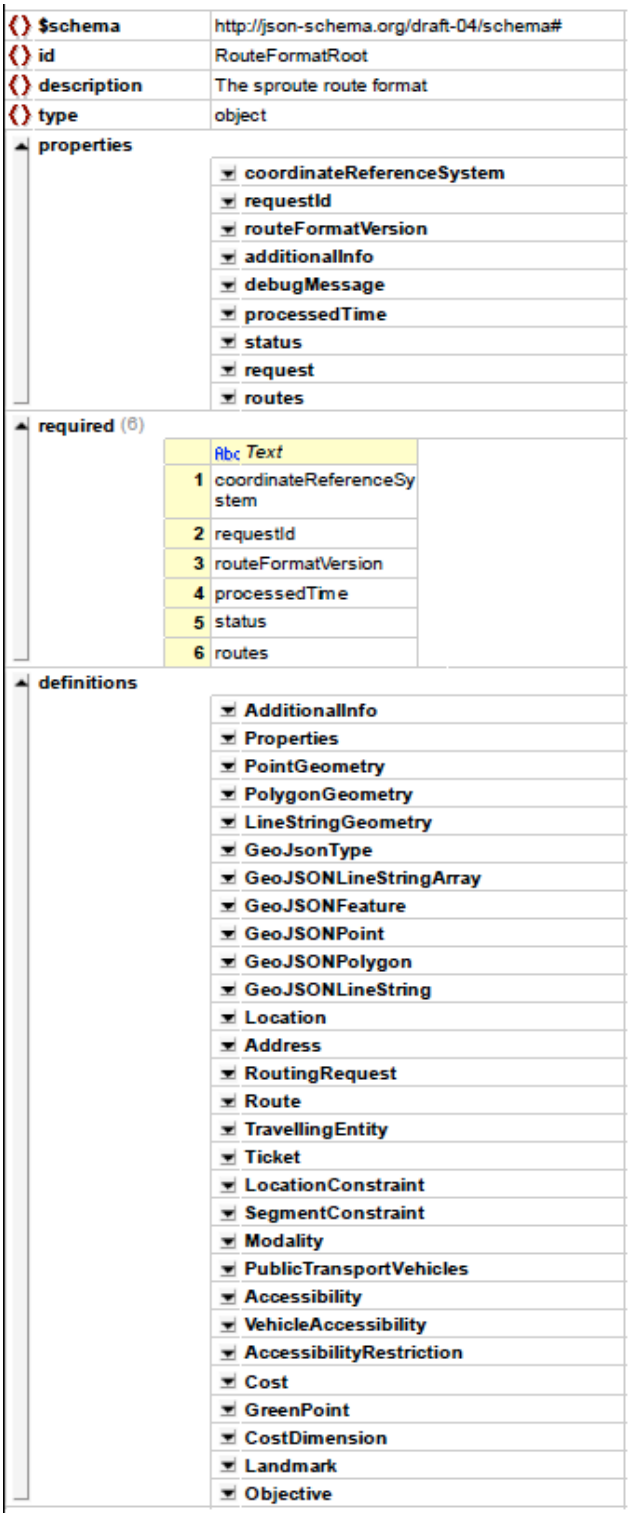

Figure 23: Fundamental concepts and structure of SPROUTE

4.1.7 Summary of protocol adaption

Route planning in BONVOAGE implements a set of personalisation and adaption functionalities. A set of extensions was required for the routing format to support the full functionality of the service. The backwards-compatible changes of the SPROUTE format[3] were described in the above sections. New concepts were added to provide more detailed route requests such as travelling entities and route leg constrains. To personalise the routing service additional information were required that were added to the format (such as optimisation objectives, tickets and real-time routing support). The route response object was extended and refined in term of costs of a trip, trip references and personalised green points.

4.2 Adaptation of an orchestrator as a service

As described in deliverable D4.1, the concept of the orchestrator is a decomposition approach to solve the routing problem on a multimodal network by means of soloists. In principle, the architecture does allow for multiple orchestrators to exist concurrently, for example as national access point available in each country.

4.2.1 Maintenance of the orchestrator graph

Once an orchestrator service is notified about a new soloist engine becoming available, it must ask the soloist about its characteristics in order to insert the new service into its overlay graph (e.g., the one exemplified in Figure 24), which constitutes the part of the MMMDB that is devoted to the federated community of soloists. An overlay graph contains vertices for all transition nodes and edges connecting the vertices it is possible to travel between. In the figure the vertices are shown as the black nodes and the edges are the blue or brown lines.

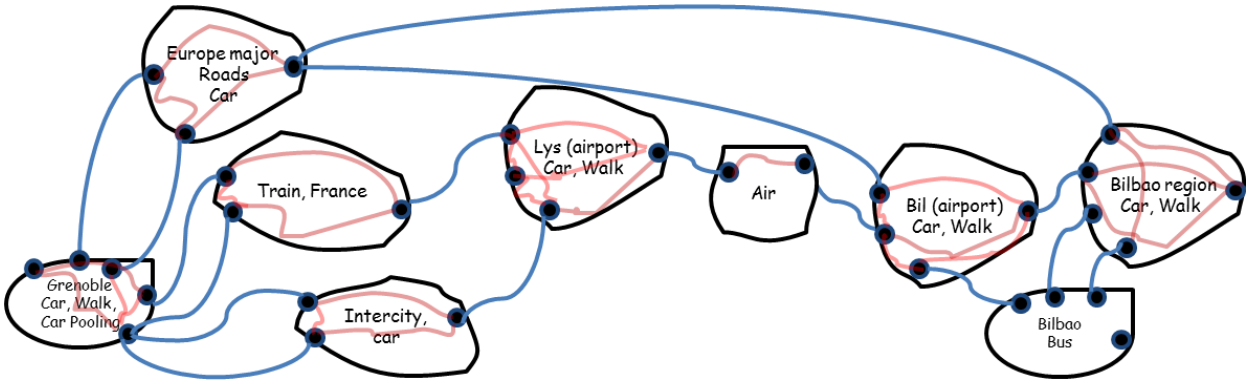

Figure 24: An overlay graph

A simple example of characteristics of a soloist was already introduced in deliverable D4.1. The schema is relying on the GeoJSON Format[4] to encoding a variety of geographic data to describe:

[3] https://github.com/dts-ait/ariadne-json-route-format
[4] The GeoJSON Format, RFC 7946, http://geojson.org/

- The geographical boundary indicates the extent of the soloist graph and the schema allows for any GeoJSON geometry object. It is normally represented as a MultiPolygon for continuous graphs (e.g., for a soloist handling road-based travels) or a MultiPoint to represent discrete positions (e.g., for a soloist handling flights between airports).
- The position of the transition nodes indicate locations where transition to other soloists can take place can also be described by means of for any GeoJSON geometry object, e.g., a Point for the location of a bus stop or a Polygon for a larger terminal area.

Figure 25 shows the classes of the soloist characteristics that are relevant to build the orchestrator overlay graph.

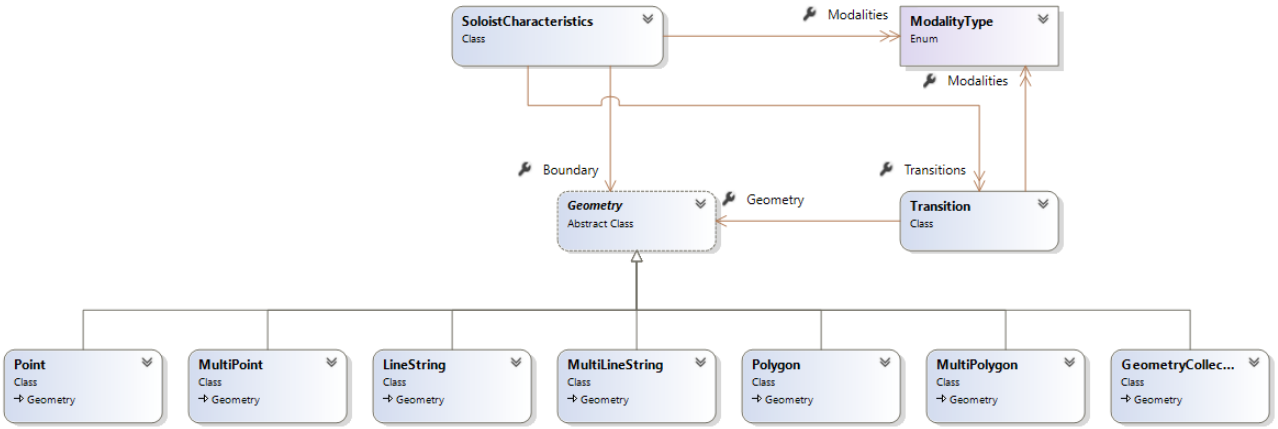

Figure 25: The characteristics of a soloist used to build the overlay graph

Vertices in the graph are created from:

- Each transition node of each soloist
- The intersection area between the boundaries of any pair of soloists with a common subset of modalities

Edges in the graph are created between:

- All vertices within the same soloist
- Vertices in different soloists from transition nodes with a common subset of modalities

Whenever the BVDS notifies the orchestrator about new soloists or removed soloists the orchestrator will update the overlay graph.

4.2.2 Handling of travel requests

When an orchestrator receives a travel request, a shortest path algorithm is applied on an overlay graph. The results of this algorithm correspond to at least one set of soloist that together can find a path from origin to destination. Requests are then issued to the respective soloists. When all soloists are finished, the responses from the soloists are merged into a single response to the travel requests. In Figure 26 an example of this is given, where the path through the overlay graph goes through multiple soloists. This in turn corresponds to the requests to the soloists. In the following sections, we describe further details on the issues pertaining to using the overlay graph.

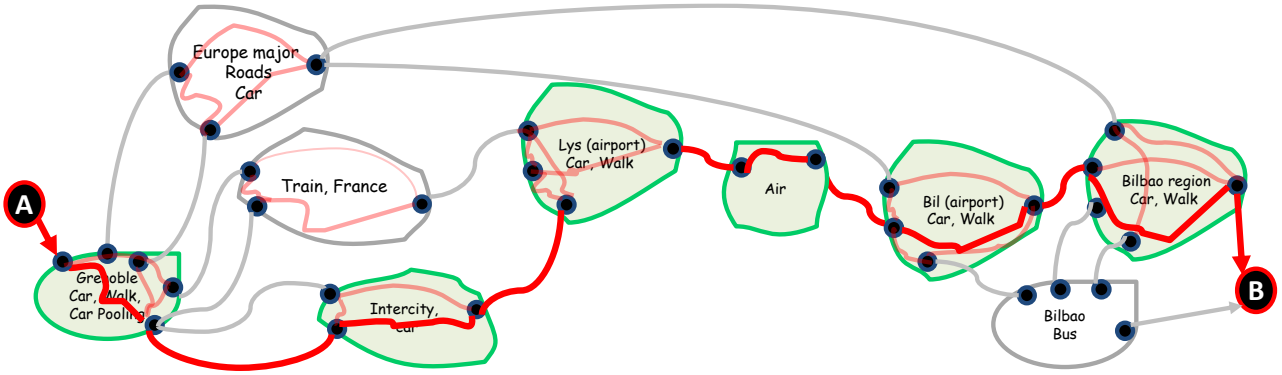

Figure 26: An overlay graph with a possible path for a travel request

4.2.3 Learning soloist performance

For each request, the orchestrator needs to identify the most promising soloists to find the *k* best routes. A learning mechanism for the lower- and upper bounds for each cost dimension is needed since not all soloists will provide the correct calculation of the travel objective. Each individual request is represented as a vector of weights per cost dimension. The lower bound for each arc (a single number) is then easily computed by multiplying the lower bound for each cost dimension times the weight vector.

Furthermore, a lower bound to the cost dimensions are in most cases correlated such that the lowest travel time will usually not be possible to obtain at the lowest price. To address this issue a clustering is performed. In this way we can get lower- and upper bounds for each cluster of users. After clustering, the bounds will then be maintained for a number of common representative weights of the cost dimensions. This gives a much better estimate on the actual objective value of the request.

To learn about the soloists' behaviour, statistics are collected over time. Statistics are gathered in two main categories, response issues and quality of solutions. The response issues are all related to how fast the soloist respond to a request, reliability and such. The quality of solutions are measured in multiple ways, first it can be tracked how often the suggested trip will become a part of an orchestrator solution, but also user feedback on the route is maintained. This is tracked in order to avoid soloists with poor quality solutions continuously been selected ahead of better solutions that might appear worse. As an example, one can imagine a soloist that ensures that public transit connections can be made, whereas another soloist sets asides inadequate time to these connections. The last task in the learning is to maintain the lower- and upper bounds, by updating them on a continuous basis when new requests are being issued, but also update them by, if necessary, through sampling of additional "synthetic" requests to the soloists.

4.3 Adaptation of a soloist as a service

In this section it is described how a soloist can act as a service. Basically two different kinds of soloists can be handled, soloist that are tailored and developed specifically to become a soloist or an external trip planning solver that is wrapped to become a soloist. How a soloist works internally is not essential as long it is able to communicate through the SPROUTE format or be mapped to the format. Further details can be found in the following sections.

4.3.1 Representation of the characteristics of a soloists

In the OGB the soloists are registered with basic information on how to discover the soloist, in this section we describe the additional characteristics describing the soloist service to the orchestrator. A soloist is characterised in a number of different dimensions, at first it must communicate to the orchestrator what geographical area it covers. The area is described as a polygon given by a set of coordinates. What modalities it is possible to handle, the objective dimensions that are supported, if the solved is static or able to handle real-time routing. Furthermore, it must be able to communicate its connection points. These are the characteristics that are static information to be communicated once. With respect to lower bounds of the soloist, the soloist should either be able to provide lower bounds or the orchestrator must establish these by issuing a number of requests to approximate the lower bounds.

4.3.2 Initialization and maintenance of the topology

As we have seen, when a soloist is connected to an orchestrator the orchestrator requires aggregated information about the travel topology of the soloist in order to update the orchestrator graph to include the soloist, and stores the graph in the MMMDB. This is mainly the connection points that are to be exchanged such that the orchestrator knows at which relevant connection points the soloist can create routes from and to and which modalities it can handle at these points.

The topology of a solver can change either by new roads or other infrastructure being opened or closed, or due to real-time events. If a soloist experiences major changes to the topology, this may again be communicated to the orchestrator in order to keep the orchestrator graph updated.

4.3.3 Wrapping of trip planning solvers as soloists

Existing journey planning websites or services can also be included as a BONVOYAGE service, e.g., for a transport service provider to promote its services or for a city to integrate its travel network into a continent-wide transport system.

As an example of how this can be done, we have wrapped the Google Maps Directions API[5] into a soloist service. The API can be access the RESTful Web services through an HTTP interface, with requests constructed as a URL string. For a directions request, the following parameters are required:

Parameter	Google Maps Directions API	SPROUTE data member
The API key	key	
To specify the output format	outputFormat (json, xml)	(using json)
To specify the location for the start of the trip	origin	from
To specify the location for the end of the trip	destination	to

Table 2: Required parameters for the Google Maps Directions API mapped to SPROUTE

In addition following optional parameters are relevant to further specify the desired route:

[5] The Google Maps API is found at https://developers.google.com/maps/documentation/directions/

Parameter	Google Maps Directions API	SPROUTE data member
To specify the transportation mode to use	mode	modesOfTransport. generalizedType
To specify additional locations the trip must go via	waypoints	via
To specify the number of trip alternative in the response	alternatives (true of false)	(using true)
To specify restrictions to avoid	avoid	accessibilityRestrictions
To specify the output language	language	Language
To specify unit system	units	(using metric)
To specify the desired time the end of the trip	arrival_time	arrivalTime
To specify the desired time the start of the trip	departure_time	departureTime
To specify assumptions to use when calculating time in traffic	traffic_model (best guess, pessimistic, optimistic)	(using best guess)
To specify the permitted travel modes allowed during a transit trip	transit_mode	modesOfTransport. detailedType
To specify travel preferences	transit_routing_preference	segmentConstraints. objectives

Table 3: Optional parameters for the Google Maps Directions API mapped to SPROUTE

As can be seen from the previous sections, most of the parameters of the API can be mapped to SPROUTE. However, when the Google API is used in a soloist service, we must map the opposite way. When doing so we encountered a number of challenges due to several features represented in SPROUTE that cannot be represented in the Google API. An example of this is the personalized travel preferences, i.e., the travel objective. Google will normally suggest alternative fastest routes, while the user may want a cheap and eco-friendly route. Due to such limitations, existing journey planning websites may not be applicable for all types of travel requests. We can deal with this in two ways:

1. The orchestrator may choose to ignore certain soloists for certain types of personalized travel preferences
2. The orchestrator may try to compute the correct travel cost as well as possible for the missing cost dimensions.

Figure 27: Three alternative routes returned from a BONVOYAGE soloist mapping the Google Maps API.

5 Adaptation of real time travel data

Providing the travel planner and final users updates in real time about transit data greatly enhances the quality of the service. Much work has been devoted during the last several years in defining clear standards for real time travel data, either by extending the existing standards for static data or by creating specific ones.

Most of them are now mature and travel operators are starting to package live updates about their fleet and services, choosing GTFS Realtime (an open extension to GTFS), DATEX II or SIRI.

Regardless of the format, the major impediment which is slowing down the diffusion of live updates of travel data is, in our opinion, the lack of a distribution mechanism able to deliver those updates in a narrowly-targeted and incremental fashion to the interested users.

The current approach is simply to update the relevant feeds, and rely on the user's capability to perform periodic downloads of the updated (and extremely bulky) files. This is obviously largely inefficient.

The most suitable paradigm is publish/subscribe, where the travel operator regularly produces the updated feed, and publishes it, while subscribers (e.g. soloists) receive it upon declaring interest in the specific feed.

Although the "periodic blind download" strategy is to be avoided at all costs, practically, also the above ideal pub/sub scenario must face the following challenges:

- given the massive amounts of live travel data streams, the operator should put up a costly network infrastructure dedicated to pushing live contents to potentially millions of subscribers.
- Subscribers would want to receive incremental updates versus being flooded with a constant stream of bulk feeds that are 90% identical to the previous one.
- Different kinds of subscribers will want to subscribe to selected portions of a whole feed: a large-scale travel planner (a soloist dedicated to train planning for the entire Spain, for instance) may want to receive live incremental updates of train schedules from multiple train operators with a nationwide spatial coverage, while a person travelling by car may want to receive road accident and traffic camera feeds for a very narrow area around him.

The following sections explain how BONVOYAGE tackles the above challenges. The solution we have designed is illustrated by first describing its application to road real time feeds packaged in DATEX II format by Norway's NPRA. The same solution is then applied to bus real time schedules of Bilbao City. This approach allows us to exemplify the flexibility of our design.

5.1 Publication and subscription to data for road travel

5.1.1 Introduction to DATEX

DATEX[6] is a file format and it represents one of the outcomes of the standardization process related to Delivering European Transport Policy[7]. In the road sector, the DATEX standard was developed for information exchange between traffic management centres, traffic information centres and service providers and it is now considered the reference for applications that have been developed in the last decade.

To support sustainable mobility in Europe, the European Commission (EC) has promoted the development of information exchange and coordination between actors of the road traffic management. The European Transport Policy and EC's ITS Action Plan have been the key for traffic information centres to cooperate. The overall activity does not only foresee collecting information, as it envisions companies and Public Administrations working together to use these information to elaborate strategies that can grant high level quality and continuity of service. This ambitious objective is in line with the EasyWay programme, a stakeholder organisation in charge of maintaining the DATEX specifications (now available in its 2nd generation version), which has been created to make roads safer and for better traffic management. The most obvious result of this process is the prevention of congested situations.

DATEX II is now covered by European Study 5 (ES5), currently chaired by Germany, which is composed of 2 Working Groups (WG): the Strategic Group (SG) and the Technical Group (TG)[8]. The former steers the work programme and is in charge of reporting to the EasyWay Steering Committee and the European Commission. After receiving terms of reference from the SG, the latter consists of experts that deal technical specification management, thus including user support and feedbacks. It is also in charge of reporting back on its progress and works in cooperation with CEN TC278 Working Group 8[9].

Technically speaking, DATEX II models relevant to the description of dynamic data about road situations are composed of three parts: the modelling methodology (which goes under the name of Context and framework - Part 1), Location referencing (Part 2) and Situation Publication (Part 3). The latter is most widely used for traffic information messages as it manages the Variable Message Sign (VMS) system.

5.1.2 DATEX II data from Norway NPRA server

At the time of this writing (September 2017, halfway through the entire project), the BONVOYAGE platform has incorporated DATEX II data referring to Norway, exposed by NPRA (Norwegian Public Road Administration), that is the Norwegian national company in charge of planning, building, operating and maintaining national and county roads. NPRA (partner of the BONVOYAGE consortium) collects weather, traffic, and any other useful information from different data sources (such as sensors on the road, weather monitoring stations, CCTVs) trying to ensure that all

[6] http://www.datex2.eu/

[7] http://www.datex2.eu/content/datex-background

[8] http://www.datex2.eu/content/datex-organisation-sg-tg-user-forum

[9] http://www.datex2.eu/content/standardization

those who walk, cycle, travel by car or use public transport get to their destination safely. Then, it exposes the collected data, divided on six DATEX II files, through the server available on:

https://www.vegvesen.no/ws/no/vegvesen/veg/trafikkpublikasjon

Specifically, the following files can be downloaded:

1. GetSituation.xml - Reference url is /trafikk/1/GetSituation. It stores the current status (and a history of events happened in the past) of roads in Norway. Available data cover (i) Road Or Carriageway Or Lane Management, (ii) Road Maintenance, (iii) Rerouting Management, (iv) Temporary Speed Limit, (v) General Network Management, (vi) Speed Management and (vii) Construction Works.
2. GetMeasuredWeatherData.xml - Reference url is /vaer/1/GetMeasuredWeatherData. It provides information regarding weather conditions. Data are automatically collected by weather stations along the road network. Available data cover (i) Road Surface Condition Information, (ii) Wind Information, (iii) Precipitation Information, (iv) Humidity Information, (v) Temperature Information, (vi) Visibility Information.
3. GetMeasurementWeatherSiteTable.xml - Reference url is /vaer/1/GetMeasurementWeatherSiteTable. It provides a reference table for all sensors available in every location mapped and provides geographical specifications;
4. GetCCTVSiteTable.xml - Reference url is /kamera/1/GetCCTVSiteTable. It provides closed circuit camera pictures for every installation, that is also geo-referenced. Available data cover (i) camera site, (ii) publication details.
5. GetTravelTimeData.xml - Reference url is /reisetid/1/GetTravelTimeData. It provides information about variation on travel time needed specified roads and paths. Data collected are (i) default travel time, (ii) road info.
6. GetPredefinedTravelTimeLocations.xml - Reference url is /reisetid/1/GetPredefinedTravelTimeLocations. In a similar way to what happens for Weather Data, these files are providing information GIS-formatted information about roads and paths.

This DATEX II data is highly dynamic and changes over time. The available approach, so far, however, forces the user to periodically download all the files from the remote web server (i.e., the NPRA web site in our example), process them, identify what portions of the file have changed since last download, and extract information of interest.

Furthermore, one should generally assume that the average user (either human or Application) is interested to data confined to a specific geographical area, making the filtering even more difficult when he is forced to download data in huge bulks. The analysis reported in the following figures demonstrates quantitatively how the whole process may clearly benefit from a distribution system able to publish only the incremental differences that happen in a confined region.

The following Figure 28 shows that the size of the file for each kind of available information does not change, on average, during the course of several days. This is expected because each file covers a huge territory, and deltas tend to compensate. We also notice how massive the files are, especially those reporting the ever-changing road situations (circa 10 Megabytes).

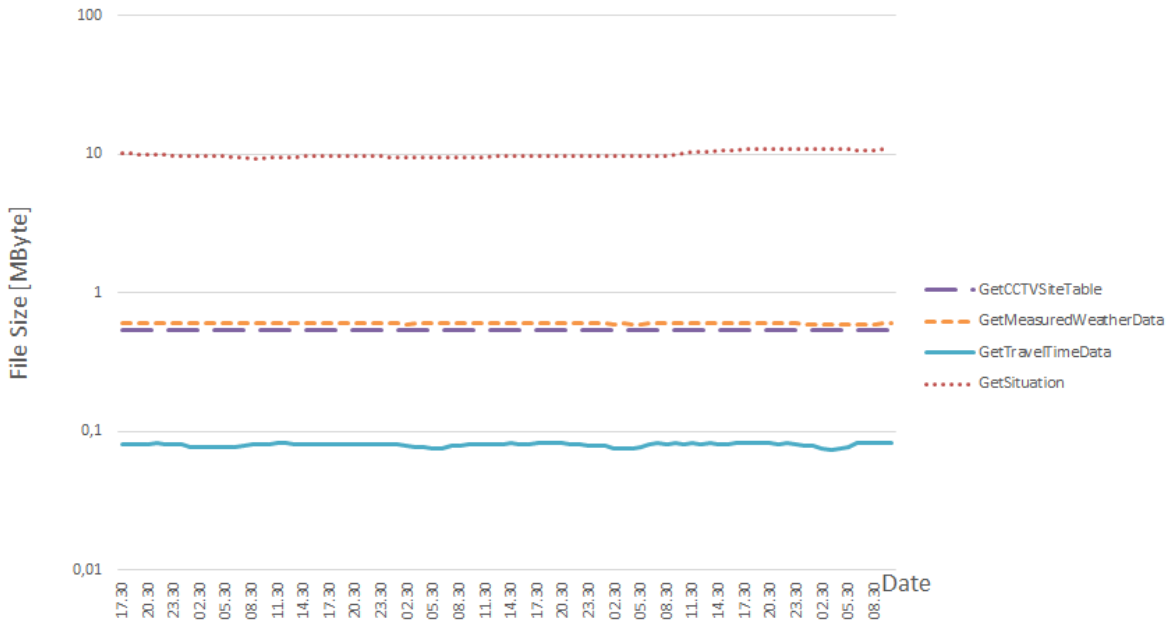

Figure 28: Size of DATEX II files over the course of several days.

More important is to notice the incremental (or delta) variation over time of the data, i.e. the percentage of each file that changes from one download to the successive one. Downloads are spaced by a time of three hours (see Figure 29).

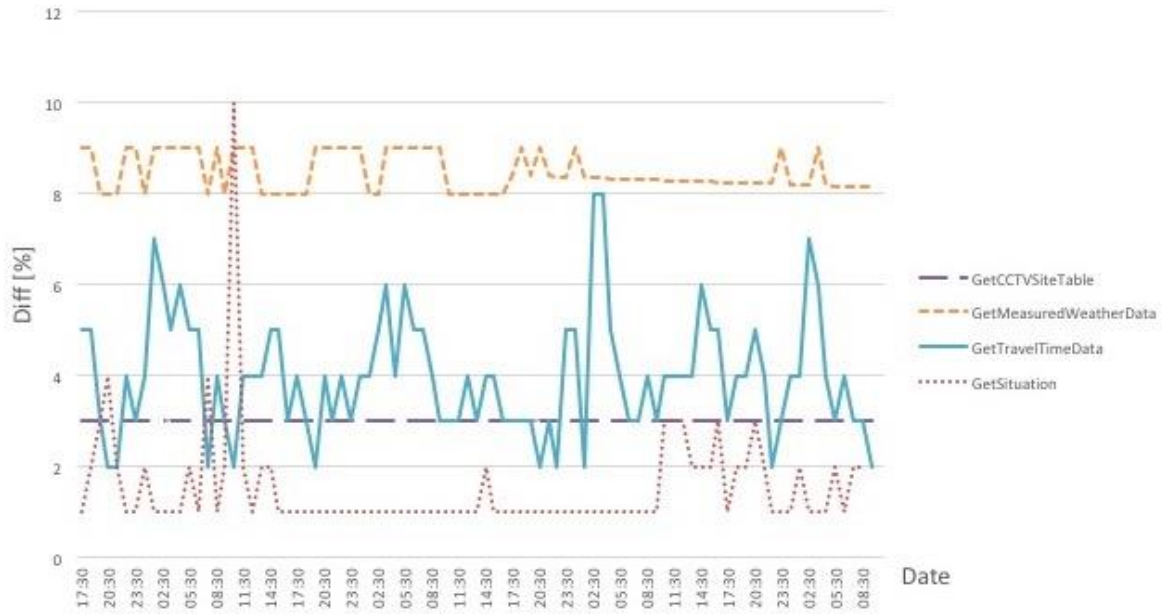

Figure 29: Percentage difference between two consecutive downloads of the same DATEX II file.

It is evident that only small changes are injected into the data over small time scales, so that downloading the same file wholly over and over is largely sub-optimal.

We now turn our attention to the internals of the data. It is important to note that each record available in DATEX II files is geo-referenced (see for instance the following figures, which display DATEX II fragments).

```
<groupOfLocations xsi:type="Linear">
  <locationForDisplay>
    <latitude>59.590786</latitude>
    <longitude>11.105595</longitude>
  </locationForDisplay>
  <locationExtension>
    <locationExtension>
      <roadNumber>E18</roadNumber>
      <countyNumber>1</countyNumber>
      <municipalityNumber>124</municipalityNumber>
    </locationExtension>
```

Figure 30: geographical information details of a specific "situation".

```
  </supplementaryPositionalDescription>
  <linearExtension>
    <linearLineStringExtension>
      <gmlLineString srsName="EPSG:4326">
        <coordinates>11.112797083071316 59.5882075876211, 11.112371446391842
        59.58844207256969, 11.111819014109614 59.58872277130089, 11.111330781607549
        59.58896024684658, 11.111075827111932 59.58907843700832, 11.110541532571752
        59.589310944454226, 11.110007694793003 59.58952566346405, 11.109434703151853
        59.589738586494214, 11.108855628933835 59.58993802128466, 11.108488184677311
        59.59005681639864, 11.107650063642172 59.590299243130346, 11.107031058734153
        59.590460533430885, 11.106323658849249 59.590630337612126,
        11.105727187659518 59.59075929923604, 11.105072754130688 59.590892645924164,
        11.1043216909902 59.59103974348769, 11.102398633604821 59.59137820237314,
        11.101471828379873 59.59152382392242, 11.10034350199191 59.5916901321551,
        11.098455894960969 59.591950029975784, 11.097104873607963
        59.592125332676034</coordinates>
      </gmlLineString>
    </linearLineStringExtension>
  </linearExtension>
</groupOfLocations>
<complianceOption>mandatory</complianceOption>
```

Figure 31: GPS coordinates indicating a series of points, namely "group of locations"

The above <latitude> and <longitude> tags (or <coordinates>) allow for a straightforward splitting of each big file into separate records that can be grouped based on their position on a map. Further, it allows for the changes of the data over time to be narrowed down to the area where they occur, enabling selective dissemination of just the portions of the file that have changed.

The following sections outline the design of a highly efficient architecture for dissemination of dynamic road situations based on the BONVOYAGE Communication System's publish/subscribe capabilities and on the OGB-based Discovery service: it offers the possibility to **subscribe to a specified rectangular region of the map** by making a geo-referenced

query, retrieving only a selection of DATEX II data exposed by the remote server, and collecting future updates in terms of small incremental differences with the previous chunk.

5.1.3 Delivery of DATEX II data through the BONVOYAGE Communication System

5.1.3.1 How to identify contents

The first step has been the design of a convenient name space for the DATEX II information at hand. This type of travel centric content is best identified through a unique name, designed according to a hierarchical and geo-referenced structure, that is:

[NAME] = /bv/[coords]/GPS-ID/[std]/[provider]/[service]/[… other fields …]

where:

(i) [coords] provides GPS coordinates of the geographical area which the content refers to,
(ii) [std] field is "datexii",
(iii) [provider] indicates data source (i.e., NPRA),
(iv) [service] field identifies the specific service the files is related to. Services identifies the macro category of DATEX II topics (e.g. traveltime, getsituation, weather and camera).
(v) [other fields] better specify required information and further details.

We decompose the space into a uniform grid; a tile of the grid can be iteratively split into smaller tiles, realizing a multi-layer hierarchical grid structure. The spatial grid is aligned with world parallels and meridians, identified by GPS longitude and latitude expressed in decimal degrees, e.g. 12.51133E 41.89193N. A level-0 tile of the grid contains all points having the same longitude and latitude value up to the dot, e.g. the level-0 tile (12,41) contains all points whose longitude and latitude start with 12 and 41, respectively. A level-*n* tile of the grid, where *n*=1,2 contains all points having the same latitude and longitude value up to the *n-th* decimals, e.g. the level-2 tile (12.51,41.89) contains all points whose latitude and longitude start with 12.51 and 41.89, respectively. We observe that each tile of level-*n* is formed by 10 tiles of level-(*n*+1). Consecutive Level-0 tiles represent increments of 1 degree latitude and longitude; moving at level-1 tiles represent increments of 0.1 degrees, and so on. Moreover, for latitude values close to the equator, the area of a level-0 tile approximates a square of 100x100 km, level-1 a square of 10x10 km and a level-2 tile a square of 1x1 km.

In summary, each level-*n* tile is identified by a tile-prefix whose structure is:

/bv/lng(0)/lat(0)/lng(1)lat(1)/.../lng(n)lat(n)/GPS-ID

where lng(0) and lng(k) are the integer and the *k-th* decimal value of GPS longitude of the tile. Same holds for latitude values. E.g. the tile (12.51,41.89) has the name prefix /bv/12/41/58/19/GPS-ID.

Please notice that the lng(0) and lat(0) values are decoupled by a slash since they may be composed by a variable number of digits, thus parsing is simpler by separating.

If we adopt a three level spatial grid, for example, the name /bv/8/61/45/36/GPS-ID/datexii/npra/getsituation identifies all the records exposed by NPRA in GetSituation.xml, that refer to a selected geographical area. The specific area is a 1 km x 1 km approx. square, with latitude ranging from 61.56 degrees to 61.57 degrees and longitude ranging from 8.43 to 8.44**.**

In general, a geographical area can be described by a set of tiles, each one identified with a unique name like the above one. Each tail may (or may not) contain DATEX II data.

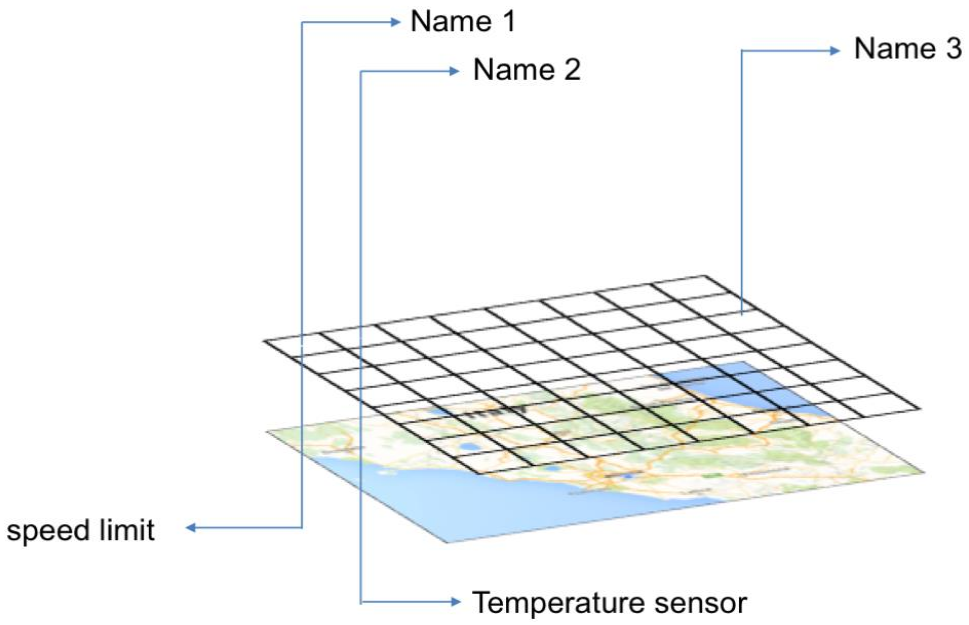

Figure 32: a set of tiles and associated data, for a geographical area.

5.1.3.2 Processing and publishing of DATEX II files by the producer

We exemplify the dissemination of real time feeds as an asynchronous Producer/Consumer interaction, which we decouple in the time dimension by means of a topic-based publish/subscribe approach. The producer publishes continuous updates under a unique name of a "topic channel". As soon as consumers subscribe to the topic, they receive the associated updates. Subscribers can subscribe to a topic even before any publisher has started publishing information for that topic. At subscription time, the subscriber decides whether to receive an initial feed comprising the whole history of publications up to the current point in time. From that point onwards, the subscriber receives differential updates pushed by the publisher under the selected topic.

The producer process is divided into two logical steps:

- MDHT processing the updated data in order to make it discoverable and to link it to a topic name
- Publisher publishing of updates under their dedicated topic name

In the first step, the original DATEX II files is processed by the MDHT. The main activities it performs are:

1. Fetching the feed from Norwegian Public Roads Administration's (NPRA) external server through the regular https link. This phase implies a direct connection to the server hosting the above-described files; it is important to underline here that the producer is the only client connecting to the external server: it is not necessary for the operator to provision a server able to support millions of clients.
2. Downloading items, identifying fields of interest and unpacking them into separate records. The MDHT, as usual, performs extraction of the geo-tagged information of each record;
3. Per-tile grouping of the records. This operation aggregates data belonging to the same tile;
4. Creating of a topic name based on the tile the records belongs to;
5. Contacting the OGB APIs to create one object for each tile, with associated properties that link the tile to the topic name belonging to it.

The following Figure 33 shows how a GeoJSON object is created for each tile, as a consequence of parsing a specific DATEX II feed, for instance the one specific to road situations, i.e. GetSituation.xml, in its entirety. The example shows that each tile has a unique topic name associated to it (for the specific data type), so that **subscriptions to the tile's** data are enabled, in order to get constant updates of all data (of that type) that insist on the tile.

This approach creates topic channels that are quasi-static over time, because unique channel names are associated to the tiles and not to the single records within the tiles. The subscriber will receive all data of the tile, which means it can receive several records, one for each new situation that has appeared in the reference tile since the GetSituation.xml feed was last updated by NPRA. It is up to the subscriber to post-filter the records based on other criteria (such as the actual coordinates of each received record).

The figure shows two distinct records (please notice their different <situationRecord id>) **falling within the same tile**. They give birth to the same GeoJSON object representing the tile and its associated topic channel name "/bv/10/59/79/70/GPS-ID/datexii/npra/getsituation", which stands for a 0.01 degrees wide tile (a level-2 tile, approx. 1 km x 1 km) with top-left longitude 10.77 and latitude 59.90. The name of the topic channel is associated to a **URL property** because it represents, in a sense, the **network contact point that the subscriber must use** to get in touch with the producer of the data, analogous to the URL property we have used in the case of static GTFS or NeTEx feeds. The Polygon represented in each GeoJSON covers exactly the 0.01 degrees wide square tiles, and its coordinates, accordingly, change at their second digit after the decimal dot.

The name includes the information that road GetSituation type of data are published under that topic channel (the "getsituation" part of the name). A different GeoJSON object is going to be created for the closed camera circuit pictures feed GetCCTVSiteTable.xml that insist on the same tile, for example, and the name is associated to it is different, i.e. "/bv/10/59/79/70/GPS-ID/datexii/npra/camera".

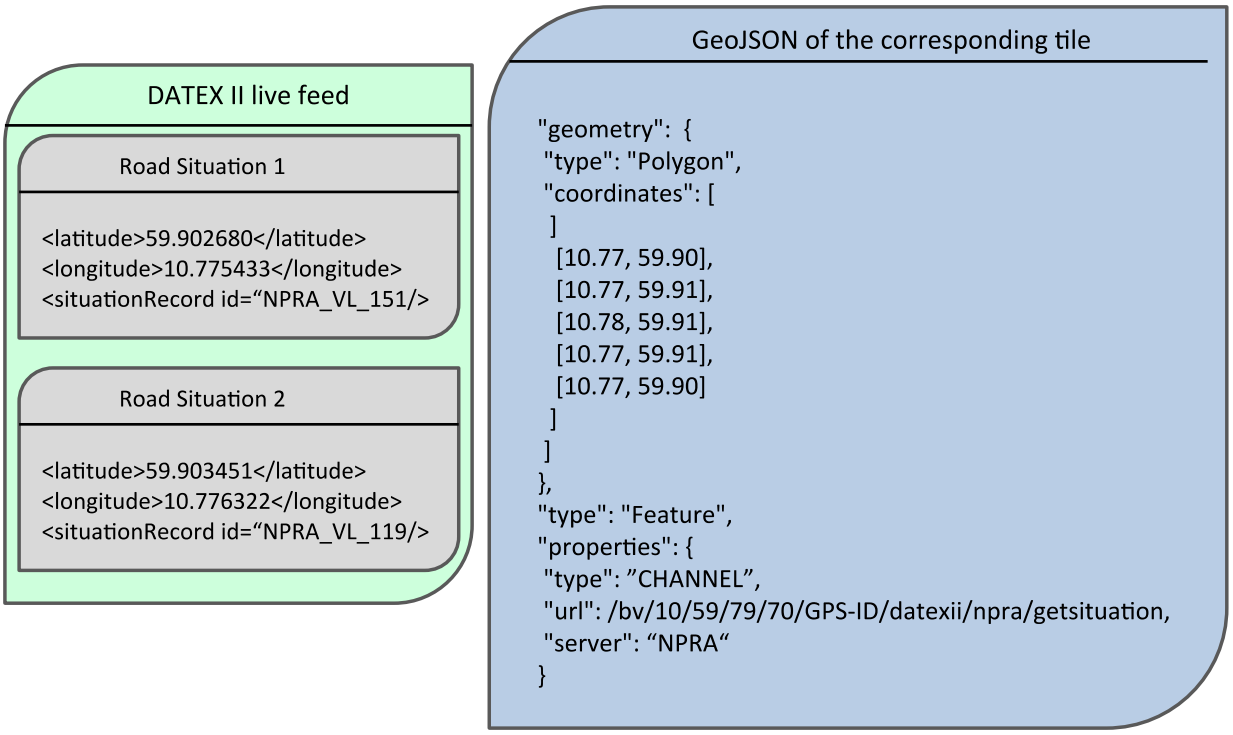

Figure 33: two DATEX II situations fall under the same 1 km x 1 km square tile.

In the second step the producer is going to publish all newly appeared records as a set of repackaged, smaller DATEX II files, under the pertinent channel, by means of the Publish primitive of the BONVOYAGE Communication System (see D3.1); Specifically, the following operations are accomplished:

1. Monitoring of the DATEX II feed on the external server, for detection of changes;
2. Creation of a new syntactically correct smaller DATEX II file for each tile that undergoes a change;
3. Announcement of availability of new items to the topic;
4. Waiting for requests from the clients to serve;

5.1.3.3 Subscribe mechanism at the consumer

The consumer process is, likewise, divided into two logical steps:

- discovering what topics are available in a selected region of interest;
- subscribing to a subset of those topics, based on filtering criteria applied to the topic name.

In the first step, a user needs to discover what topics are available in the rectangular region of interest for him. For instance, if the user is a nation-wide soloist engine, the region would cover all of Norway, otherwise, if the user is a human, it may be a small area around the person sitting in a car or standing nearby a bus stop.

Discovery is accomplished, just like it was the case for static GTFS data feeds discovery, by issuing a spatial query to OGB, i.e. by invoking the OGB.rangeQuery() API. As we have seen, the OGB.rangeQuery must be passed a geographical area of interest, identified by a GPS point and a size in degree. For example this may result in a rectangular area comprised between the two following GPS points (expressed in decimal notation): (Longitude 6°, latitude 63°)-(Longitude 12°, latitude 58°), which identifies the following area of interest:

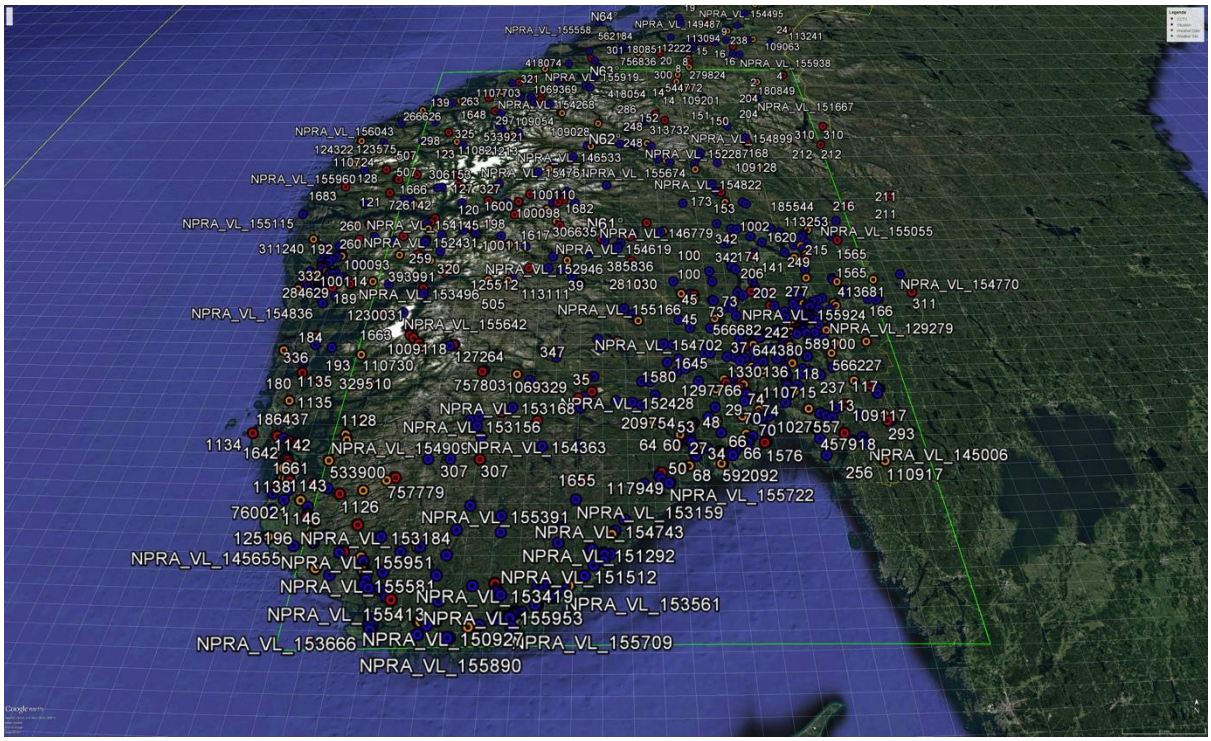

Figure 34: Selected area of Interest, showing the density of GetSituation records.

Figure 34 qualitatively shows what is the density of a typical sample of DATEX II GetSituation records in the above-mentioned area. Depending on the size of the tiles, one or more records can fall within one single tile. As a reference point, it is interesting to notice here that the entire surface of Norway is going to be divided into 10900 tiles of 0.1 degrees wide area (approx. 10 km x 10 km, that is level-1 tiles, which have just two levels in the name hierarchy, contrary to the three levels of the names we have employed in the example figures of the previous sections. Level-2 tiles would be 100 times more dense).

As a result of the discovery process, a set of topics is given back to the user. The user must now filter them based on their name, excluding names that contain, for instance, components referring to types of data he is not interest in subscribing to.

At the second step the consumer process passes to the BONVOYAGE Communication System as many names as wanted, asking for subscriptions to the topics they represent. Specifically:

1. For each name, a subscription request is created and handed to the BONVOYAGE Communication System.
2. The BONVOYAGE Communication System searches for available contents for each topic.

3. Available records referring to the identified names are retrieved.
4. As soon a new record is available, the BONVOYAGE Communication System delivers it back to the subscriber.

For sake of clarity, an example of the retrieved contents with the level-1 name:

/bv/06/58/63/GPS-ID/datexii/npra/situation,

is provided in Figure 35 and Figure 36. They represent just one of the several records received as a consequence of the subscription to the above topic. The first picture shows the initial instance of the record.

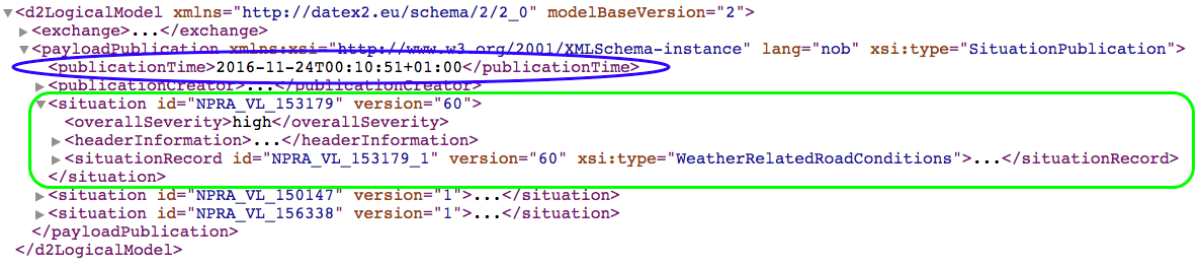

Figure 35: First instance of a record received via subscription to a 0.1 degrees wide tile

The following picture shows that an update for the same record was received 5 hours later.

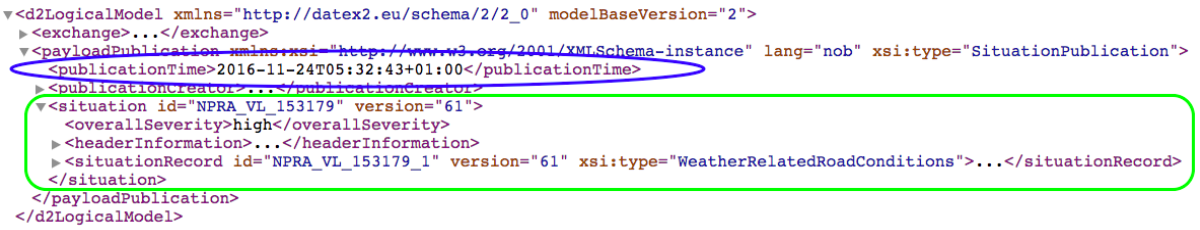

Figure 36: Update received some time later

5.2 Publication and subscription to data for public transport

SIRI is an XML protocol to allow distributed computers to exchange real-time information about public transport services and vehicles. SIRI is based on the Transmodel abstract model for public transport information, a general-purpose model, and XML schema for public transport information.

Although SIRI is the ideal candidate for exchanging real-time information of PT, in BONVOYAGE we don't have any partner hosting SIRI-formatted data. Hence we focused first in managing the information available at the City of Bilbao servers.

5.2.1 Bilbao real time data about bus timetables

City of Bilbao has implemented, within the Co-Cities project, the so-called Commonly Agreed Interface platform that uses WFS (Web Feature Service) as the standard for access to geo-localized information about transport in the city area. The Open Geospatial Consortium Web Feature Service Interface Standard (WFS) that was adopted in Bilbao provides an interface allowing requests for geographical features across the web, using platform-independent calls. Generally speaking, the WFS specification defines interfaces for describing data manipulation operations of geographic features. Data manipulation operations include the ability to:

- get or query features based on spatial and non-spatial constraints
- create a new feature instance
- delete a feature instance
- update a feature instance

The basic Web Feature Service allows querying and retrieval of features.

The static interface contract for the OGC Web Service is as follows: when writing a WFS, one must implement the following operations:

- GetCapabilities - this queries the WFS service to determine available options. It allows to get a list of data from the server, as well as operation WFS and their parameters.
- DescribeFeatureType - this retrieves the XML schema to allow the WFS client to parse the resultsets. It allows getting information related to a specific data set.
- GetFeature - this performs the actual query - parameters such as bounding box and any other filters should be passed in, as appropriate, and the WFS service then returns a GML resultset containing full geometry and feature attributes. This is basically to get data, including its geometry and values of the attributes.

All operations should include at least three parameters: service, version and request.

The above static interface is the one implemented in Bilbao, which supports the following data:

- Static public transport:
  - ept:Route
  - ept:StopPoint
  - ept:Service
  - ept:Line
- Dynamic public transport:
  - ept:ProductionTimetable
  - ept:DatedTimetableVersionFrame
  - ept:JourneyPattern
  - ept:PointInJourneyPattern
  - ept:EstimatedCall
- Static parking:
  - edi:ParkingPoint
- Dynamic parking:
  - eti:CarParkDynamic

- Traffic:
  - enw:RoadNetLink
  - enw:RoadNetNode
  - eti:Activities
  - eti:ConstructionWorks
  - eti:GeneralObstruction
  - eti:MaintenanceWorks

In order to showcase how the BONVOYAGE Communication System can be used to efficiently disseminate dynamic information about **public** transport, in this section we will describe the approach that was taken to distribute updates of arrival times of different bus lines of Bilbao.

This section contrasts with the previous ones, which are focused on disseminating information about **road** transport and dynamic road situations.

The main difference is that with public transport a key human-readable name is generally readily available when referring to the transport itself, be it the bus **line name** or the bus **stop name** the user wants to receive information about.

Thus, in order to experiment with different possible designs for adapting existing services to the BONVOYAGE architecture, in this section we describe an approach where the **namespace has been constructed around the name of the public transport**.

As an alternative to the schema proposed in the previous section, these types of travel centric contents can be identified through a unique name, designed according to a hierarchical but **non-georeferenced** structure, that is:

[NAME] = /bv/[provider]/[transport_type]/[transport_identifier]/[service]

where:

(i) [provider] indicates data source (i.e., bilbao),
(ii) [transport_type] field identifies the specific macro category of transports (e.g. bus, metro),
(iii) [transport_identifier] is the name that identifies what specific transport the user is asking information for (e.g. the name of a metro line or the name of a bus stop)
(iv) [service] specifies the kind of information requested (e.g. timetables)

For example, the name /bv/cityofbilbao/bus/zubileta/timetables identifies all the records exposed by the City of Bilbao servers that have information about bus stops named “zubileta”, specifically regarding timetables.

5.2.2 Real-time bus timetables parsing

The following query gives access to all currently available Calls about arrival times of busses of a certain Line at a certain Stop. The feature of interest is **EstimedCall**.

<wfs:GetFeature xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wfs http://schemas.opengis.net/wfs/1.1.0/wfs.xsd" xmlns:gml="http://www.opengis.net/gml" xmlns:wfs="http://www.opengis.net/wfs" xmlns:ept="http://namespace.emotion-project.eu/version/Final2.1.0/pubtrans" xmlns:ogc="http://www.opengis.net/ogc" service="WFS" version="1.1.0" > <wfs:Query typeName="**ept:EstimatedCall**"></wfs:Query> </wfs:GetFeature>

A typical call to the above service returns the following information (please notice the result presented here is already stripped of all tags that are note relevant to the discussion):

<EstimatedCall id="EstimatedCall:1">

<LocalisedCharacterString>PLAZA BIRIBILA</LocalisedCharacterString>

<LineCode>10</LineCode>

<aimedArrivalTime>2017-02-08T11:10:58.080Z</aimedArrivalTime>

<StopPointCode>1101</StopPointCode>

</EstimatedCall>

Unfortunately, a typical call to the above service for obtaining the realtime arrival time of a bus on average returns more than 2 megabytes of data, that is thousands of EstimatedCall records such as the one above. From the perspective of a single user, continuous polling of the service in order to get fresh information about a specific trip is unfeasible. With the above call she gets all available information for all stops.

The user is generally interested in arrival times at a specific bus stop, which she is able to identify either by the bus stop name or the bus stop position.

The following query gives access to all StopID and their coordinates. Stops are the basic anchor for our georeferencing. The feature of interest is **StopPoint**.

<wfs:GetFeature xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wfs

http://schemas.opengis.net/wfs/1.1.0/wfs.xsd" xmlns:gml="http://www.opengis.net/gml" xmlns:wfs="http://www.opengis.net/wfs"

xmlns:ept="http://namespace.emotion-project.eu/version/Final2.1.0/pubtrans" xmlns:ogc="http://www.opengis.net/ogc" service="WFS" version="1.1.0" > <wfs:Query typeName="**ept:StopPoint**">

</wfs:Query>

</wfs:GetFeature>

A typical answer to the StopPoint call looks like:

<StopPoint id="2">

<pos>-2.9228838481644965 43.26624131429908</pos>

<StopPointCode>2201</StopPointCode>

<stopPointName>Anselma de Salces (2)</stopPointName>

<shortName>Anselma de Salces (2)</shortName>

</StopPoint>

By joining the dynamic information returned by the EstimatedCall and the static information returned by the StopPoint calls, our producer process is able to:

- Step 1: use the MDHT to generate a set of topic names for the set of existing bus stops of Bilbao. The set of names is static, and a GeoJSON object is inserted in OGB for each stop, representing it as a Point with an associated URL (that is the generated name). These Point objects play exactly the same role as the tiles do for DATEX II.
- Step 2: Continuously monitor the EstimatedCall data stream and publish all newly detected EstimatedCall each under the topic (i.e. bus stop name) they belong to.

Likewise, for the consumer process it is now extremely easy to:

- Step 1: discover what bus stop names are present in an area of interest
- Step 2: use those names as topics for firing network subscriptions in the BONVOYAGE Communication System, and receive notifications of bus timetable updates (EstimatedCall) as soon as they occur.

In this case, the flexibility of being able to publish the stream of updates under a topic that consists of a human-readable name of the stop (or of the metro line, for instance) allows the user to subscribe simultaneously to all updates that belong to that stop, regardless of the bus numbers (or to the updates belonging to the metro line).

The metro line is thus considered as the indexing entity, able to join all data that belongs to it **without splitting the line across several geographic tiles.**

5.2.3 Aviation travel data

With *aviation travel data*, we mean the information necessary to include travels with aircrafts as part of an overall intermodal trips, for example:

- Name and geographical position of airports
- Flight schedules
- Travel fares (i.e., tariff schemes) offered for single- or combinations of flights
- Real time delays and cancellations of flights

We have used the openflights.org as the source for identifying the IATA and ICAO code names and the geographical position of airports. They provide information about almost all airports across the globe available for download as plain csv files.

Worldwide aviation flight schedules are not yet made openly available in a standard format, but some commercial sources such as oag.com, amadeus.com and google.com provide API's for flight schedules that can be used for trip planning services. In our current BONVOYAGE aviation soloist, we have chosen to use data from the Google QPX Express API, which allows requests and responses in JSON format. In the request we must specify a specific origin airport, a destination airport, a date, and a passenger category (i.e., adult, child, senior). The response will then contain all possible trip options between the given airports at the given date. Each single trip option describe a sequence of one or more flights and one or more pricing options. The data model of Google QPX Express is shown in Figure 37.

Access to real time delays and cancellations of flights would allow us to realize a trip planning soloist service dedicated to monitor planned flights with respect to how deviations affects the subsequent parts of an overall intermodal trip. For example, real-time position and velocity of aircrafts is available from the OpenSky Network (opensky-network.org). The OpenSky Network is a community-based receiver network which continuously collects air traffic surveillance data and provides a JSON-based REST API to track. However, we have not found a way to process this information into information about delayed arrival times, as would be needed for a potential BONVOYAGE real-time aviation soloist service. On the other hand, information about expected delays and cancellations of European flights are also collected and distributed by the EUROCONTROL Network Manager. We have found that this information is only made available to organisations which are actively engaged in aircraft operations and related support services. We are therefore currently only able to realize pre-trip aviation trip planning. As data sources concerning real-time delays and cancellations later may be made available, on-trip control and re-planning can easily be realized.

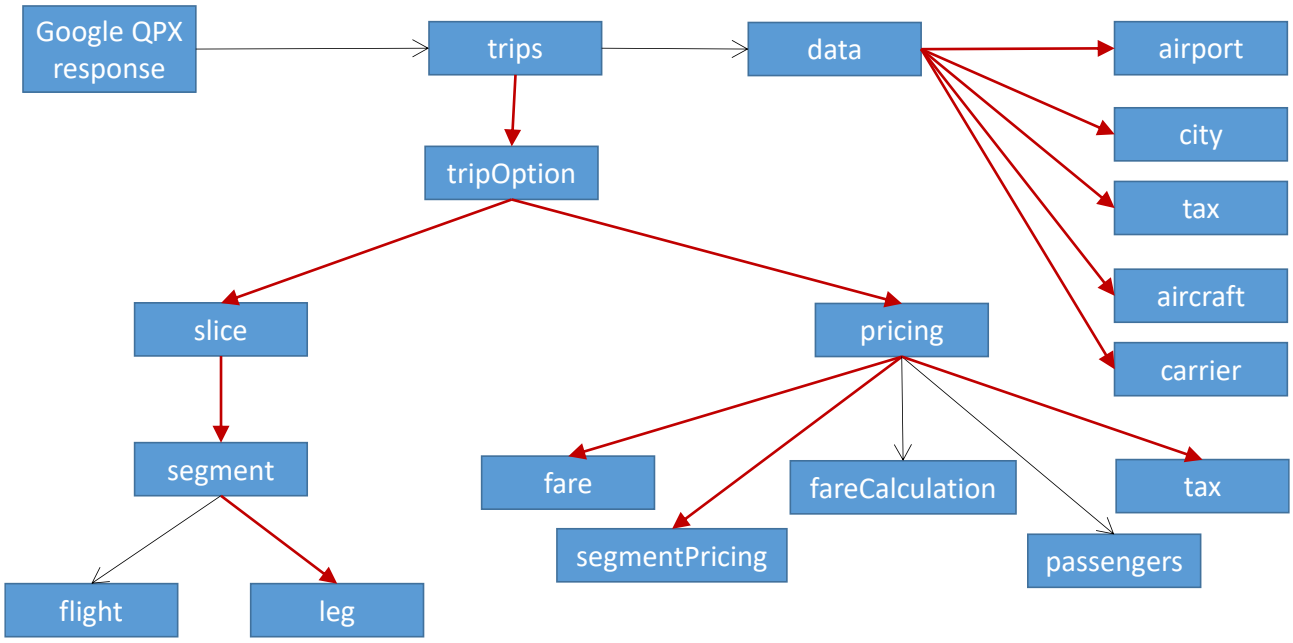

Figure 37: The data model used by the Google QPX Express API

6 Conclusion and future work

The work described in this deliverable was focused on the design of service adaptation solutions. The approach we present was, of course, validated by means of preliminary implementation of said solutions, but the core implementation work is still on-going. The upcoming deliverable D5.2 will document this implementation work.

We must notice here that, although the distinction between design and implementation is clear, in this Work Package we witnessed a very strong feedback between implementation and refinement of the designed solutions. For instance, the overall schema of how to tackle in a unitary way the adaptation of both static and dynamic data sources, as well as external resources such as the Soloists, clearly emerged only after we had developed several prototypal adaptation components specific to the different data we wanted to include. In other words, in this Work Package the bottom-up design strategy was maybe more prominent than in other WPs.

Deliverable D5.2 will follow along these lines and will have a complete and deeper account of the effectiveness and performance of the solutions we have presented here.