

SYNCHRONICITY



GA no:	732240
Action full title:	SynchroniCity: Delivering an IoT enabled Digital Single Market for Europe and Beyond
Call/Topic:	Large Scale Pilots
Type of action:	Innovation Action (IA)
Starting date of action:	01.01.2017
Project duration:	36 months
Project end date:	31.12.2019
Deliverable number:	D3.2
Deliverable title:	Suite of baseline implementations - basic
Document version:	1.0
WP number:	WP3
Lead beneficiary:	35-NEC
Main author(s):	Flavio Cirillo, Detlef Straeten (NEC), Jose Gato, David Fernandez Gomez (ATOS), Giuseppe Ciulla (ENG), Ignacio Elicegui (UC), Jaime Ventura (POR), Antti Nurminen (FVH), German Molina Carrasco, Ruben Molina Moreno (HOPU), Paolo Fosci (MIL), Arturo Medela (TST)
Internal reviewers:	Luis Muñoz (UC), Gemma Ginty, Reza Akhavan (FCC), Martin Brynskov (AU)
Type of deliverable:	Other
Dissemination level:	PU
Delivery date from Annex 1:	M16
Actual delivery date:	M20 (30.08.2018)

This deliverable is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732240.



Co-funded by the
European Union

Executive Summary

This document has the aim of depicting the outcome of two tasks: first, the identification of baseline services and, secondly, their reference implementation, which comprises the design, architecture and specification.

Target Audiences for this deliverable are exploiters of baseline services, e.g. RZs application developers, third parties joining from the Open Calls, and external users of SynchroniCity.

In a nutshell, the main objectives tackled in this deliverable can be outlined in the following list:

- To stand out the functional and non-functional requirements related to the SynchroniCity application themes, extracted from (SynchroniCity, D3.1, 2018).
- To study and analyze a first set of baseline services
- To provide replicable reference implementations of these services
- To define the way they should be implemented and distributed
- To settle down the foundations for other implementations, either for the second version of this deliverable (D3.3), or other participants, like the ones coming from the Open Calls

Before digging into more technical aspects, it is worth defining the concept of baseline-service that we will rely on henceforth. Despite a baseline service can be seen as a very abstract concept, in the scope of Synchronicity, we understand it as **a single functional block using data input to return any kind of feature, either managing, enriching, joining or filtering the input data from SynchroniCity infrastructure, and/or data coming from other sources**. Other specific characteristics of a SynchroniCity's baseline-service are outlined below:

- They are built in top of the SynchroniCity framework (SynchroniCity, D2.1, 2017) and exploit the different interaction points offered by the platform, e.g. NGSI interfaces and APIs (OMA, 2012), historical data collection (ETSI NGSI-LD, 2018) and open data (DCAT-AP, 2015), Data Marketplace API¹ (this last one will be addressed in the future).
- Therefore, data input will be NGSI-compliant (in terms of data models (SynchroniCity, D2.2, 2018)) so as to respect OASC²'s principles. Consequently, interoperability and replicability will be guaranteed.
- The outcome generated by these services must carry out a concrete (and focused) functionality over the input data
- An API to interact with the service and obtain this extra functionality.
- Enabling the development of other applications and services, including those ones addressed in T3.3 (SynchroniCity, D3.5, 2018), and opening the door to new integration possibilities for the Open Calls.
- Enabling replicability and demonstrating the Synchronicity framework and OASC principles. It is envisioned that these services could be straightforwardly downloadable and deployable in different environments.

The idea of a standalone and fully-fledged functionality accessible via API (web service) matches pretty well with the widespread concept of micro-service; but, in the scope of Synchronicity there is no obligation to implement the services following typical micro-service's architectures concepts such as integration bus.

¹ <https://projects.tmforum.org/wiki/display/API/Open+API+Table>

² <http://oascities.org/>

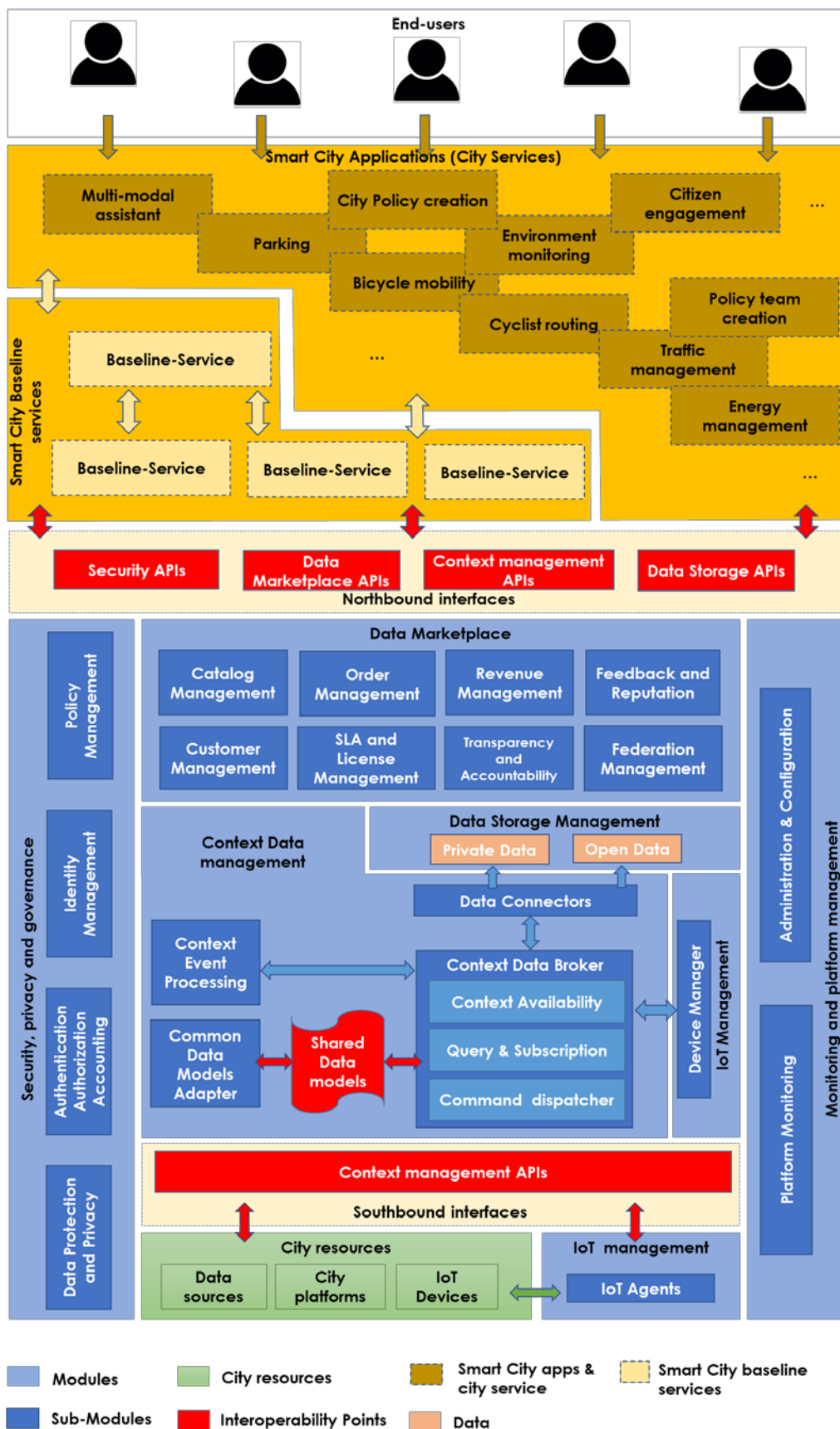


Figure 1 Full stack of the SynchroniCity Reference Architecture from sensors and data sources to end users (citizens). It is comprehensive of the baseline and city services layer on top of the SynchroniCity northbound interface

One of the main macro-objectives of SynchroniCity is to pilot IoT-enabled services that directly address citizens. With that view, SynchroniCity is offering **city services** that are applications built for citizen targeting citizens' needs with the aim of "smartify" the life within urban area. As support for the city services, **baseline services** are offering facilities for the one hand easing their development and reduce the time-to-market and on the other hand share efforts among city services teams for a co-creation approach.

A baseline service is therefore seen as a generic software enabler that is compatible and easily integrated with the reference architecture specified in WP2 for the targeted applications and services. This approach is towards the digital single market for IoT-enabled smart cities in Europe and beyond. It is expected that the three SynchroniCity application themes (defined in (SynchroniCity, D3.1, 2018)) and Open Calls will be good candidates to make use of these baseline services to ease its development, enable replicability and demonstrate OASC principles and the SynchroniCity framework advantages.

The identification of the baseline services has started from the outcomes of T3.1, documented in (SynchroniCity, D3.1, 2018). In that document common use-cases and functional and non-functional requirements are identified for each application theme with no differentiation between end user services and baseline services. From the presented requirements list, every Reference Zone (RZ) assessed its concrete priority for each Functional Requirement. This approach has suggested the boundary between the functional requirements likely to be addressed by shared baseline services and other higher value application services, in case a feature was requested by 2 or more RZs. Other RZ specific requirement will be addressed at application development and application customization in T3.3.

The work of identification of baseline services is on-going and it is strictly intertwined with the work of applications design within T3.3. In addition all the consortium partners involved in T3.2 are offering their expertise to identify added value common features to applications which might concretize in new shared services identification. Moreover new baseline services might come from the third parties of the Open Calls that will be issued during the SynchroniCity lifetime.

In a parallel work, available assets from the consortium partners and/or the market have been analyzed to find matches with the FRs in order to maximize the re-use of what is already available and established in the market.

This deliverable is to be considered as an intermediate milestone and it will be extended in the second and final issue of Baseline Services, deliverable D3.3.

Abbreviations

API	Application Programming Interface
CSV	Comma Separated Values
D	Deliverable
EC	European Commission
FR	Functional Requirement
GTFS	General Transit Feed Specification
OTP	Open Trip Planner
REST	Representational State Transfer
RZ	Reference Zone
WP	Work Package
WT	Work Task

Contents

Contents

EXECUTIVE SUMMARY	1
ABBREVIATIONS.....	4
LIST OF FIGURES.....	8
LIST OF TABLES.....	9
1 INTRODUCTION	11
1.1 COMMON FUNCTIONAL REQUIREMENTS (FRs) AMONG RZs.....	12
1.2 FUNCTIONAL REQUIREMENTS (FRs) ADDRESSED BY SYNCHRONICITY INFRASTRUCTURE	16
1.3 DISCLAIMER.....	20
2 BASELINE SERVICES.....	21
2.1 BASELINE SERVICES RESPONSIBILITIES AND ADOPTION MAP	21
2.1.1 RESPONSIBILITIES MAP	21
2.1.2 ADOPTION MAP	22
2.2 ROUTING BASELINE SERVICE.....	23
2.2.1 DESCRIPTION.....	23
2.2.2 APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	23
2.2.3 API.....	24
2.2.3.1 Information Model	30
2.2.3.2 Interaction diagram (with the external world).....	32
2.2.4 SERVICE ARCHITECTURE	33
2.2.4.1 Operation model (Message flows between components).....	34
2.2.4.2 Non-functional requirements.....	35
2.2.5 SERVICE IMPLEMENTATION	36
2.2.5.1 Service implementation IT environment (dependencies like SQL db)	36
2.2.5.2 Service deployment dependencies (e.g. Orion Broker, oAuth server etc.).....	36
2.2.5.3 Installation and admin guide	36
2.3 PARKING PROBABILITY ESTIMATION	39
2.3.1 DESCRIPTION.....	39
2.3.2 APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	39
2.3.3 INTERFACE	39
2.3.3.1 Parking estimation API	39
2.3.3.1.1 Data Structures.....	43
2.3.3.2 Information Model	44
2.3.3.3 Interaction diagram (with the external world).....	47
2.3.4 SERVICE ARCHITECTURE	49

2.3.4.1	Service components	49
2.3.4.2	Operation model (Message flows between components).....	51
2.3.4.3	Non-functional requirements.....	53
2.3.5	IMPLEMENTATION ROADMAP	53
2.4	TRAFFIC FLOW ESTIMATOR BASELINE SERVICE	55
2.4.1	DESCRIPTION.....	55
2.4.2	APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	55
2.4.3	INTERFACE	55
2.4.3.1	Traffic Service API.....	55
2.4.3.2	Information Model	58
2.4.3.3	Interaction diagram (with the external world).....	59
2.4.4	SERVICE ARCHITECTURE	62
2.4.4.1	Service components	62
2.4.4.2	Operation model (Message flows between components).....	63
2.4.5	IMPLEMENTATION ROADMAP	64
2.5	SMARTCITIES DASHBOARD SERVICE	67
2.5.1	DESCRIPTION.....	67
2.5.2	APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	67
2.5.3	INTERFACE	68
2.5.3.1	Dashboard functionalities	68
2.5.3.2	Backend server API.....	70
2.5.3.3	Information Model	71
2.5.3.4	Interaction diagram (with the external world).....	71
2.5.4	SERVICE ARCHITECTURE	71
2.5.4.1	Service components	72
2.5.4.2	Operation model (Message flows between components).....	73
2.5.5	SERVICE IMPLEMENTATION	73
2.5.5.1	Service implementation IT environment (dependencies like SQL db)	73
2.5.5.2	Service deployment dependencies (e.g. Orion Broker, oAuth server etc.).....	73
2.5.5.3	Installation and admin guide	73
2.6	METRICS VISUALIZER (GRAFANA) SERVICE	74
2.6.1	DESCRIPTION.....	74
2.6.2	APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	74
2.6.3	API.....	75
2.6.3.1	Interface	76
2.6.3.2	Backend server API.....	76
2.6.3.3	Information Model	76
2.6.3.4	Interaction diagram (with the external world).....	76
2.6.4	SERVICE ARCHITECTURE	76
2.6.4.1	Service components	77
2.6.4.2	Operation model (Message flows between components).....	78
2.6.5	SERVICE IMPLEMENTATION	78
2.6.5.1	Service implementation IT environment (dependencies like SQL db)	78
2.6.5.2	Service deployment dependencies (e.g. Orion Broker, oAuth server etc.).....	78
2.6.5.3	Installation and admin guide	78
3	CONCLUSIONS AND FUTURE DEVELOPMENTS	79
BIBLIOGRAPHY.....		81

List of Figures

Figure 1 Full stack of the SynchroniCity Reference Architecture from sensors and data sources to end users (citizens). It is comprehensive of the baseline and city services layer on top of the SynchroniCity northbound interface	2
Figure 2. The web interface of the generic OTP service.	33
Figure 3. Development view of OTP.	34
Figure 4. Functional view of OTP: how a request is handled.	35
Figure 5 ParkingSpot, OnStreetParking and OffStreetParking relations	47
Figure 6 List Parking Areas	48
Figure 7 Parking Area Detail	48
Figure 8 Free Parking Spots Estimation.....	49
Figure 9 Parking probability estimator service architecture and interaction with the SynchroniCity framework.....	50
Figure 10 Parking estimator implementation roadmap (basic service).....	53
Figure 11 Traffic flow estimator baseline service interaction diagram	60
Figure 12 List Traffic Flow	60
Figure 13 : Current Traffic Flow Conditions.....	61
Figure 14 Traffic Flow Conditions Estimation	61
Figure 15 Traffic flow estimator service components	62
Figure 16 Traffic Flow Estimator service message flow	64
Figure 17 Gantt diagram of the Traffic Flow Estimation roadmap envisaging 10 weeks to achieve a first full implementation, without considering testing	66
Figure 18 SmartCities Dashboard view.....	67
Figure 19 Advanced panel for visualizing device information.....	70
Figure 20 SmartCities Dashboard architecture	72
Figure 21 Grafana Dashboard example	74
Figure 24 Interaction Between Services.....	76
Figure 25 Metrics visualizer architecture	77
Figure 26 Metrics visualizer architecture	77

List of Tables

Table 1 Human-centric traffic management application theme Functional Requirements prioritized by the Reference Zones	13
Table 2 Multi-modal transportation application theme Functional Requirements prioritized by the Reference Zones.....	15
Table 3 Community Policy Suite application theme Functional Requirements prioritized by the Reference Zones	16
Table 4 Human-centric traffic management application theme Functional Requirements already addressed by the SynchroniCity framework.	17
Table 5 Multi-modal transportation application theme Functional Requirements already addressed by the SynchroniCity framework.Green coloured tick is for a full fulfilment whereas a yellow coloured tick for a partial fulfilment.	19
Table 6 Community Policy Suite application theme Functional Requirements already addressed by the SynchroniCity framework.Green coloured tick is for a full fulfilment whereas a yellow coloured tick for a partial fulfilment.	20
Table 7 Responsibilities o of the services developed. Dark red coloured cells refer to the head responsible of the services, light red cells indicate other member of the services team.....	21
Table 8 Adoption map between services and applications developed by RZs.....	22
Table 9 Multi-modal transportation application theme FRs addressed.....	24
Table 10 Complete list of GET parameters for the Routing Baseline Services API.....	29
Table 11 Multi-modal transportation application theme FRs addressed.....	39
Table 12 List Parking Areas API	41
Table 13 Parking Area Detail API.....	41
Table 14 Free Parking Spot Estimation API.....	42
Table 15 ErrorResponse Entity	43
Table 16 OnStreetParking/OffStreetParkingParkingArea Entities	44
Table 17 ParkingSpot Entity	45
Table 18 OnStreetParking Entity	46
Table 19 OnStreetParking Entity	47
Table 20 Multi-modal transportation application theme FRs addressed.....	55
Table 21 List Traffic Flows API.....	57
Table 22 Current Traffic Flow Conditions API	57
Table 23 Traffic Flow Conditions Estimation API	58
Table 24 ErrorResponse Entity	58
Table 25 TrafficFlowObserved Entity	59
Table 26 TrafficFlowConditionEstimation Entity	59
Table 27 FRs of the Human-Centric Traffic Management application theme addressed by the Smartcities Dashboard baseline service	68
Table 28 FRs of the Multi-Modal Transportation application theme addressed by the Smartcities Dashboard baseline service.....	68
Table 29 FRs of the Community Policy Suite application theme addressed by the Smartcities Dashboard baseline service	68
Table 30 FRs of the Human-Centric Traffic Management application theme addressed by the Metric Visualizer (Grafana) baseline service	75

Table 31 FRs of the Community Policy Suite application theme addressed by the Metric Visualizer (Grafana) baseline service.....	75
Table 32 Percentage of addressed FRs in common between RZs	79
Table 33 Human-Centric Traffic Management application theme FRs already addressed	82
Table 34 Multi-Modal Transportation application theme FRs already addressed.....	84
Table 35 Community Policy Suite application theme FRs already addressed.....	85

1 Introduction

This deliverable presents the result of Task 3.2 which targets the identification and implementation of baseline services that will be used as building blocks for the implementation of applications and city services.

The pilot SynchroniCity's city services are categorized in three application themes (SynchroniCity, D3.1, 2018):

- *Human-centric traffic management* has the goal to improve bicycle mobility in cities, in a balanced way. By leveraging and combining data from different IoT systems data-driven bicycle mobility aims to improve the overall cycling experience, safety, infrastructure planning and policy making for the future.
- *Multi-modal transportation* has as the key goal to integrate public transport with other modes as well as with new mobility services based on the sharing paradigm. It aims to provide city travelers with an attractive alternative to the use of private vehicles (in particular, cars), embracing also the Mobility as a Service (MaaS) paradigm that blends public and private means of transportation and providing a single-access point to manage trips.
- *Community policy suite* has the objective of enabling local authorities to become more agile in their management and policy making processes (Agile Governance). Because IoT provides real-time information, this can not only be used in a pre-defined way for operational day-to-day management but can provide an information and intelligence layer for city governance which is currently not being utilized to its full extent. As cities identify new problems or initiatives, the use of accurate, timely data sources, and their combination to provide intelligence for complex challenges, can accentuate innovation. To better inform the city's governance, this application theme provides means to combine IoT data with historical and static data to better inform: (a) management decisions (b) strategic decisions.

The baseline services implementations are in the form of generic software enables compatible and integrated with the reference architecture specified in (SynchroniCity, D2.1, 2017). The nature of the baseline services facilitates reference zone customization and stimulates replication based on common solutions and designs. These common solutions prevent both "vendor lock-in" from the cities' perspective and "city lock-in" from the perspective of city service and middleware providers. All the baseline services are delivered in a way that are easily customized by following the methodology and using the tools presented in the deliverable D3.4 (SynchroniCity, D3.4, 2018).

The identification of baseline service first started from the analysis of the Functional Requirements (FRs) identified in Task 3.1 (SynchroniCity, D3.1, 2018). The analysis of the maximum set of common function requirements is depicted in section 1.1. Then in section 1.2 is reported the assessment about FRs already addressed by the SynchroniCity infrastructure (SynchroniCity, D2.1, 2017).

Next step of our analysis have been to identify available assets from technical partner that are addressing 1 or more common FRs and most likely to be re-used by more than 1 RZs (see section 2.1). From this approach, that we could define bottom-up, we have been able to address 3 baseline services: *Routing baseline service* (section 2.2), *Smartcities Dashboard service* (section 2.5) and *Metrics Visualizer service* (section 2.6).

In addition we followed a top-down approach, where from initial city services architecture (outcome of Task3.3) of RZs and/or their technical partners, needed baseline services were identified. This was the case of the *Parking Probability estimator service* (section 2.3) and the *Traffic flow estimator service* (section 2.4).

The latter approach is still ongoing and the final outcome will be reported in the next version of this deliverable D3.3. The baseline services described in the following sections are at different levels of maturity at the time of the elaboration of this report. For that reason we invite the reader to examine details about the status the implementation roadmap in the respective sub-section for each of the

baseline services. All baseline service will be at the end wrapped into a docker container following the directives of (SynchroniCity, D3.4, 2018).

1.1 Common Functional Requirements (FRs) among RZs

Deliverable D3.1 (SynchroniCity, D3.1, 2018) identifies Functional Requirements (FRs) for each of the application theme as outcome of the least common application scenarios among RZs. Those FRs need to be addressed as application. As expected there are Common Requirements among RZs but sometimes related to RZ specific scenario (constrained by data availability, RZs strategy etc.) as well. Therefore it was missing the classification between FRs to be addressed by shared effort, concretized in baseline service, and FRs to be addressed by the specific RZ or its technical partners.

In order to distinguish between those FRs a questionnaire has been issued to all the RZs with the aim to understand the RZs' priorities for each of the FRs. A choice between 6 levels of priority was given: priority 0 to 5, where 0 means that the requirement is not needed/not applicable in the RZ, 5 means that it is of the utmost importance.

From the answered questionnaires we have derived a ranked list of FRs as candidates implemented in Baseline Services with the following criteria:

- fixed a threshold to filter out those FRs that will not be considered for baseline service: more than 2 RZs with priority equal or higher than 3
- ranked by priorities average.

The result is presented in the three Table 1, Table 2 and Table 3.

Table 1 presents the Human-Centric Traffic Management application theme which is supported by Antwerp, Milan and Eindhoven. Table 2 presents the FRs of Multi-Modal Traffic Management application theme supported by Helsinki, Porto, Milan and Santander. Finally Table 3 presents the Community Policy Suite application theme supported by Manchester, Porto and Carouge.

The FRs highlighted as green are the ones to be considered to be possibly addressed within T3.2 whilst the one colored red are left to specific implementation of the interested RZs. The ranking gave us the priority on which FR has most impact when addressed as a Baseline Service. The resulting ranking is represented in the following table where the FR number and description is the one given by (SynchroniCity, D3.1, 2018) each FR has an unique id in the scope of the single application theme, the ranking are shown by RZs (Ant for Antwerp, Mil for Milan, Ein for Eindhoven, Hel for Helsinki, Por for Porto, San for Santander, Man for Manchester and Car for Carouge), and finally they are averaged:

Results for Human Centric Traffic Management:

		Ant	Mil	Ein	Avg
FR-8	The system should provide different views on the superimposed data sets	5	5	3	4,33
FR-10	The system should allow selection of specified time window and display corresponding aggregated data over the selected time window	5	4	4	4,33
FR-11	The system may provide advice/prediction based on aggregated historical data	5	4	2	3,67
FR-6	The system must provide flexibility in terms of viewing, or having access to, certain data sets	4	5	1	3,33

	(e.g. through dashboard widgets)				
FR-5	The system must provide user identification and access control at least by a username/password basic security	5	0	5	3,33
FR-7	The system may provide a map as the main way of interaction to superimpose data sets	1	5	3	3,00
FR-12	The system may allow for annotation specified time windows	3	4	1	2,67
FR-2	System should recognize preconfigured traffic situations and classify these situations in real-time	3	0	4	2,33
FR-1	System must process data from multiple sources/sensors in real-time	1	0	5	0,00
FR-3	System should select optimal traffic light decision and actuation based on classified traffic situations in real-time	1	0	5	0,00
FR-4	System should select optimal public street lighting decision and actuation based on classified traffic situations in real-time	1	0	0	0,00
FR-9	The system should provide user admin features to control their teams via admin panels	0	4	1	0,00

Table 1 Human-centric traffic management application theme Functional Requirements prioritized by the Reference Zones

Results for Multi-modal transportation:

		Hel	Mil	Por	San	Avg
FR-4	The user should be able to enter (or provide by the GPS sensor embedded in his mobile device) the departure point, destination point and optionally other intermediates points of an itinerary on a map.	5	5	5	5	5,00
FR-5	The user should be able to enter the departure point, destination point and optionally other intermediates points of an itinerary writing their addresses, GPS coordinates or Point of Interest (PoI).	5	5	5	5	5,00
FR-2	The application may provide basic features that can be accessed in an unregistered way.	5	4	5	5	4,75
FR-6	The user should be able to select among different traveler profiles (e.g.: by walk,	5	5	5	4	4,75

	drive a car, drive a bike, use public transportation, etc.).					
FR-8	The application must permit to plan routes by considering user preferences (panoramic itinerary, air quality, lowest cost, lowest time, etc.) and requirements	5	5	5	4	4,75
FR-9	The user should be able to view on a map alternative routes and select the one he/she wants to use.	5	4	5	4	4,50
FR-11	The user should be able to select the preferred route he/she want to use.	3	5	5	4	4,25
FR-13	The application may consider real-time information (deleted rides, delays, etc.) to better plan and update the journey.	4	5	4	4	4,25
FR-14	The app should notify the user in real-time about issues that affect selected route.	4	5	4	4	4,25
FR-17	The application must permit to search for car parks by considering user preferences (proximity of stations, bus stop, destination address, secured park, Park & Ride features, etc.)	2	5	4	5	4,00
FR-29	Vehicle docking stations must at least provide User information, Vehicle info, Status and their GPS location.	4	3	5	4	4,00
FR-32	Mobility premises must at least provide their GPS location and their status.	2	5	5	4	4,00
FR-7	The user must be able to combine multiple transportation modes and mobility infrastructures (e.g.: metro + bike sharing for last mile instead of bus).	3	2	5	5	3,75
FR-16	The application must provide users with a georeferenced search of parking areas.	1	5	4	5	3,75
FR-18	The application must be able to indicate which transport possibilities are currently available near the selected parking area (e.g. estimated arrival times for nearby bus lines).	2	5	4	3	3,50
FR-30	Vehicle sensors must at least provide Vehicle/transportation line info, Speed and their GPS location.	3	2	5	4	3,50
FR-31	Environmental sensors must at least provide humidity, Air quality, noise and wind speed.	5	1	5	3	3,50
FR-33	User devices must at least provide their GPS location.	1	5	5	3	3,50
FR-1	The application must provide user identification and access control at least by a username/email address and	3	2	2	5	3,00

	password basic security (log-in).					
FR-26	Traffic sensors must at least provide their GPS location and the traffic flow intensity.	2	3	5	2	3,00
FR-3	User must register an account and log-in to access features of the application that cannot be used in an unregistered way.	3	1	2	5	2,75
FR-12	The user should be able to view on a map the points of interest located in an area near to the route.	1	3	3	4	2,75
FR-15	The application should be able to consider user related information to suggest routes based on his/her preferences.	2	4	2	3	2,75
FR-19	The application should be able to indicate in real-time where free parking lots are available and at which cost/minute.	1	5	0	5	2,75
FR-20	The application may be able to estimate when and where it will be possible to find a free parking lot.	1	5	0	4	2,50
FR-21	The application may indicate alternative parking lots/areas nearby when there's no parking availability (or the parking availability estimated probability is too low) on the user-selected area.	1	5	0	3	2,25
FR-10	The user should receive the amount/estimation of the cost (e.g. total ticket price) and arrival time (when possible) of each alternative route.	3	2	3	0	2,00
FR-27	Parking sensors must at least provide their GPS location and their status.	1	3	0	4	2,00
FR-22	The application must be able to receive and track the routes, means, check-ins and check-outs by the users, to share the revenue among the different transport providers.	2	1	2	0	0,00
FR-23	The application must be able to handle different subscription forms and tariffs decided by the transport providers.	2	1	3	0	0,00
FR-24	The application may allow the user to pay a single periodic bill for all transportation services providers.	1	1	3	0	0,00
FR-25	The application may provide the user with a report of all routes and their relative costs, and the total cost.	2	1	3	0	0,00
FR-28	Ticketing devices must at least provide User information, transport mean and their GPS location.	1	1	3	0	0,00

Table 2 Multi-modal transportation application theme Functional Requirements prioritized by the Reference Zones

Results for Community Policy Suite:

		Man	Por	Car	Avg
FR-1	Hierarchical login; system should cater for login and access to different dashboards / datasets based on the hierarchy of the local authority	5	5	4,00	4,67
FR-3	Data management; system should be able to take custody of the data and edit for visualization	5	5	4,00	4,67
FR-4	UI requirements; system should be intuitive and easy to use	4	5	5,00	4,67
FR-5	User admin; authorities should be able to control their own teams / users via admin panels	5	3	3,00	3,67
FR-2	Editable dashboards; based on city / team requirements user should be able to amend and lock (based on hierarchy) the dashboard "widgets"	3	3	4,00	3,33
FR-8	Meta reporting; for statistics on data and user interactions with the system	3	3	3,00	3,00
FR-9	Workflows; for emailing up of reports and creating a level of autonomy between other systems	3	3	3,00	3,00
FR-7	Connection to SynchroniCity digital marketplace; for north and south bound data purposes	3	0	5,00	2,67
FR-6	System should be able to connect to legacy systems available in cities for data acquisition	0	0	5,00	0,00

Table 3 Community Policy Suite application theme Functional Requirements prioritized by the Reference Zones

1.2 Functional Requirements (FRs) addressed by SynchroniCity infrastructure

In case some of the common FRs are already addressed by the instantiation of the SynchroniCity framework identified in WP2 and described in (SynchroniCity, D2.1, 2017) and the adoption of SynchroniCity security framework, we de-selected those FRs from the feature list for the needed Baseline Services Find below a more detailed discussion of those excluded features by theme.

		Avg	WP2
FR-8	The system should provide different views on the superimposed data sets	4,33	
FR-10	The system should allow selection of specified	4,33	

	time window and display corresponding aggregated data over the selected time window		
FR-11	The system may provide advice/prediction based on aggregated historical data	3,67	
FR-6	The system must provide flexibility in terms of viewing, or having access to, certain data sets (e.g. through dashboard widgets)	3,33	
FR-5	The system must provide user identification and access control at least by a username/password basic security	3,33	✓
FR-7	The system may provide a map as the main way of interaction to superimpose data sets	3,00	
FR-12	The system may allow for annotation specified time windows	2,67	
FR-2	System should recognize preconfigured traffic situations and classify these situations in real-time	2,33	

Table 4 Human-centric traffic management application theme Functional Requirements already addressed by the SynchroniCity framework.

In Table 4 the credentials management of users of the Human-Centric Traffic Management application theme (FR-5) is offered directly by the security framework of the SynchroniCity architecture, where a Single Sign-On (SSO) approach is envisioned for all the city services that will be developed following the SynchroniCity philosophy. This, on the one hand, would relieve the application developer from the burden of managing credentials, and on the other hand, the city service users (e.g. citizens) to create a new user for every new city service or application.

		Avg	WP2
FR-4	The user should be able to enter (or provide by the GPS sensor embedded in his mobile device) the departure point, destination point and optionally other intermediates points of an itinerary on a map.	5,00	
FR-5	The user should be able to enter the departure point, destination point and optionally other intermediates points of an itinerary writing their addresses, GPS coordinates or Point of Interest (PoI).	5,00	
FR-2	The application may provide basic features that can be accessed in an unregistered way.	4,75	
FR-6	The user should be able to select among different traveler profiles (e.g.: by walk, drive a car, drive a bike, use public transportation, etc.).	4,75	
FR-8	The application must permit to plan routes by considering user preferences (panoramic itinerary, air quality, lowest cost, lowest time, etc.) and requirements	4,75	
FR-9	The user should be able to view on a map alternative routes and select the one he/she wants to use.	4,50	

FR-11	The user should be able to select the preferred route he/she want to use.	4,25	
FR-13	The application may consider real-time information (deleted rides, delays, etc.) to better plan and update the journey.	4,25	
FR-14	The app should notify the user in real-time about issues that affect selected route.	4,25	
FR-17	The application must permit to search for car parks by considering user preferences (proximity of stations, bus stop, destination address, secured park, Park & Ride features, etc.)	4,00	
FR-29	Vehicle docking stations must at least provide User information, Vehicle info, Status and their GPS location.	4,00	✓
FR-32	Mobility premises must at least provide their GPS location and their status.	4,00	✓
FR-7	The user must be able to combine multiple transportation modes and mobility infrastructures (e.g.: metro + bike sharing for last mile instead of bus).	3,75	
FR-16	The application must provide users with a georeferenced search of parking areas.	3,75	✓
FR-18	The application must be able to indicate which transport possibilities are currently available near the selected parking area (e.g. estimated arrival times for nearby bus lines).	3,50	
FR-30	Vehicle sensors must at least provide Vehicle/transportation line info, Speed and their GPS location.	3,50	✓
FR-31	Environmental sensors must at least provide humidity, Air quality, noise and wind speed.	3,50	✓
FR-33	User devices must at least provide their GPS location.	3,50	
FR-1	The application must provide user identification and access control at least by a username/email address and password basic security (log-in).	3,00	✓
FR-26	Traffic sensors must at least provide their GPS location and the traffic flow intensity.	3,00	✓
FR-3	User must register an account and log-in to access features of the application that cannot be used in an unregistered way.	2,75	✓
FR-12	The user should be able to view on a map the points of interest located in an area near to the route.	2,75	
FR-15	The application should be able to consider user related information to suggest routes based on	2,75	

	his/her preferences.		
FR-19	The application should be able to indicate in real-time where free parking lots are available and at which cost/minute.	2,75	✓
FR-20	The application may be able to estimate when and where it will be possible to find a free parking lot.	2,50	
FR-21	The application may indicate alternative parking lots/areas nearby when there's no parking availability (or the parking availability estimated probability is too low) on the user-selected area.	2,25	
FR-10	The user should receive the amount/estimation of the cost (e.g. total ticket price) and arrival time (when possible) of each alternative route.	2,00	
FR-27	Parking sensors must at least provide their GPS location and their status.	2,00	✓

Table 5 Multi-modal transportation application theme Functional Requirements already addressed by the SynchroniCity framework. Green coloured tick is for a full fulfilment whereas a yellow coloured tick for a partial fulfilment.

In Table 5, the FRs for the Multi-Modal Transportation application them addressed by SynchroniCity framework are not only the ones related to credentials management and login access (FR-1 and FR-3), but also the FRs related to data and metadata quality of the IoT deployment in the RZs (FR-19, FR-27, FR-29, FR-30, FR-31, FR-32, and partially FR-26) that are addressed by the integration of the IoT sensors into the SynchroniCity architecture and by the adoption of FIWARE data models (SynchroniCity, D2.2, 2018). The FR-26 is only partially addressed by the data coming from the sensors, because the installed sensors are reporting the traffic activity over the sensors, but not the traffic flow status of roads within the city, e.g. the traffic intensity going one east-west direction rather than west-east direction. This part is left open to a baseline service (see section 2.4). Finally the geographic discovery feature of the SynchroniCity reference architecture directly addresses FR-16.

		Avg	WP2
FR-1	Hierarchical login; system should cater for login and access to different dashboards / datasets based on the hierarchy of the local authority	4,67	✓
FR-3	Data management; system should be able to take custody of the data and edit for visualization	4,67	✓
FR-4	UI requirements; system should be intuitive and easy to use	4,67	
FR-5	User admin; authorities should be able to control their own teams / users via admin panels	3,67	
FR-2	Editable dashboards; based on city / team requirements user should be able to amend and	3,33	

	lock (based on hierarchy) the dashboard "widgets"		
FR-8	Meta reporting; for statistics on data and user interactions with the system	3,00	
FR-9	Workflows; for emailing up of reports and creating a level of autonomy between other systems	3,00	
FR-7	Connection to SynchroniCity digital marketplace; for north and south bound data purposes	2,67	

Table 6 Community Policy Suite application theme Functional Requirements already addressed by the SynchroniCity framework. Green coloured tick is for a full fulfilment whereas a yellow coloured tick for a partial fulfilment.

The Community Policy Suite application theme FRs addressed by the SynchroniCity framework are shown in Table 6. The part of FR-1 related to the users' credentials management is addressed by the security framework whilst the different view based on the hierarchy is left to be addressed by the services. Similarly for FR-3, where the data storage is handled by the SynchroniCity framework but the data editing for visualization is left to services.

1.3 Disclaimer

The baseline services described in the following sections are making often usage of third parties tools and services to accomplish the meant functionalities. The usage of those tools and services is not meant to be any form of advertisement of any specific tools, services, or company. Furthermore the baseline services are under the full control of the baseline services' responsible team and they are in charge of checking any license and restrictions of third parties tools and/or services.

2 Baseline services

This section describes the identified Baseline Services at the time of writing of this document. The first 3 baseline services will be mainly leveraged for the development of city services as part of the “Multi-modal transportation” application theme: the *Routing baseline service*, the *Parking probability baseline service* and the *Traffic flow estimator baseline service*. These services might be also useful for others application themes. Other 2 services have been identified as potentially useful for a) quickly browsing the data available in a SynchroniCity framework instance and b) visualize timeseries. The services are described in the following subsections 2.2, 2.3, 2.4, 2.5, and 2.6.

In section 2.1 the partners responsible for each of the identified baseline services are depicted together with the association map between baseline services and Reference Zone that will adopt them.

2.1 Baseline Services responsibilities and adoption map

Each of the baseline services described in the following subsection (from section 2.2 to section 2.6) has a defined responsible team with a team leader. Section 2.1.1 presents a view of the mapping between the teams and the baselines services. In addition, section 2.1.2 illustrates the usage for each of the baseline services by each of the application theme.

2.1.1 Responsibilities map

	Routing Service	Parking estimator service	Traffic flow estimator service	Map visualization of data service	Timeseries visualization service
ATOS					
ENG					
FVH					
HOPU					
Milan					
Porto					
TST					
UniCAN					

Table 7 Responsibilities of the services developed. Dark red coloured cells refer to the head responsible of the services, light red cells indicate other member of the services team.

Table 7 shows the responsible teams, within the consortium, for each of the service. The responsible team is the one that is leading the design and implementation of the service, the owner of already existing asset, or leading the customization of an already existing asset from third party. In any case the responsible team is the one that will be firstly contacted for supporting baseline service users, by supplying documentation, finding solutions to problem or even updating the software and documentation. A more intense red indicates the head responsibilities whereas a lighter red indicates a member of the services team. The teams' member list might change in the future due to convergence of interests or any other reasons.

2.1.2 Adoption map

		Routing Service	Parking estimator service	Traffic flow estimator service	Map visualization of data service	Timeseries visualization service
Human-centric traffic management	Antwerp					
	Eindhoven					
	Milan					
Multi-modal transportation	Helsinki					
	Milan					
	Porto					
	Santander					
Community Policy Suite	Carouge					
	Manchester					
	Porto					

Table 8 Adoption map between services and applications developed by RZs

Table 8 presents the first adoption map between applications developed by RZs and the current list of services. Already a baseline service, the Routing Service, is going to be adopted by applications across two application themes. The many white cells or rows are due to the fact that some city services will follow a monolithic approach without making usage of the baseline services, the reader is invited to consult the outcomes of Task 3.3 for more details (SynchroniCity, D3.5, 2018) (SynchroniCity, D3.6, 2019). In addition in spite of being used by no applications (such the case of the Timeseries visualization service), some baseline services are still considered part of the SynchroniCity portfolio in order to have wider offer for third parties and for future exploitation of RZs.

2.2 Routing baseline service

The Helsinki RZ Pilot utilizes the Digitransit³ platform which consists of four different online services, namely Routing, Geocoding, Map and Realtime Service. A user interface integrates these services in an independent manner using lightweight protocols. Hence, Digitransit implements a micro-service architecture. Among these services, at the time of writing this document, only the routing service has been selected as baseline services since only this service will be utilized also by Santander, Porto and Milan RZs (see Table 8).

2.2.1 Description

Routing is the main Digitransit service, with OpenTripPlanner⁴ (OTP) as the core. The routing service allows one to run queries for routes, bus and taxi stops, city bikes and bicycling routes, disruption info and do itinerary planning.

The Routing baseline service deviates from the generic OTP, as Helsinki Region Transport has modified it into a version that better accommodates the needs of Helsinki. One of these differences is the API, which for this baseline service is based on GraphGL, while the generic OTP still utilizes a more traditional REST API. More details will be presented in the following sub-sections.

2.2.2 Application theme Functional Requirement addressed

FR code	FR description	Comments
FR-4	The user should be able to enter (or provide by the GPS sensor embedded in his mobile device) the departure point, destination point and optionally other intermediates points of an itinerary on a map.	Accepted as input by the service
FR-6	The user should be able to select among different traveler profiles (e.g.: by walk, drive a car, drive a bike, use public transportation, etc.).	Accepted as input by the routing service
FR-7	The user must be able to combine multiple transportation modes and mobility infrastructures (e.g.: metro + bike sharing for last mile instead of bus).	Accepted as input by the routing service
FR-8	The application must permit to plan routes by taking into account user preferences (panoramic itinerary, air quality, lowest cost, lowest time, etc.) and requirements	Accepted as input by the routing service
FR-9	The user should be able to view on map alternative routes and select the one he/she wants to use.	Alternative route supported by the routing service

³ <https://digitransit.fi/en/>

⁴ <http://www.opentripplanner.org/>

FR-15

The application should be able to take into account user related information in order to suggest routes based on his/her preferences.

Accepted as input by routing service

Table 9 Multi-modal transportation application theme FRs addressed

2.2.3 API

Digitransit OTP Routing API provides a way to plan itineraries and query public transportation related information about stops and timetables. The API is based on GraphQL⁵, originally developed by Facebook. When implementing a GraphQL API, server developers specify a GraphQL schema that defines what can be queried. Instead of the client calling REST-like URLs, it generates different GraphQL queries and sends these queries to the single endpoint of the API. The server then parses the query, executes it, and returns the results back to client. All GraphQL queries are sent using the POST method.

Example 1: Query all bus routes where number is like "58".

Query:

```
{
  routes(name: "58", modes: "BUS") {
    id
    agency {
      id
    }
    shortName
    longName
    desc
  }
}
```

Response:

```
{
  "data": {
    "routes": [
      {
        "id": "Um91dGU6SFNM0jI10DI=",
        "agency": {
          "id": "QWdlbmN50khTTA=="
        },
        "shortName": "582",
        "longName": "Espoon keskus-Juvanmalmi-Kalajärvi-Serena",
        "desc": null
      },
      {
        "id": "Um91dGU6SFNM0jI10DJL",
        "agency": {
          "id": "QWdlbmN50khTTA=="
        },
        "shortName": "582K",
        "longName": "Espoon keskus-Koskelo-Juvanmalmi-Kalajärvi-Serena",
        "desc": null
      }
    ]
  }
}
```

⁵ <https://graphql.org/>

```
...  
  ]  
}  
}
```

Example 2: Plan an itinerary from location (60.168992,24.932366) to (60.175294,24.684855).

Query:

```
{  
  plan(  
    from: {lat: 60.168992, lon: 24.932366}  
    to: {lat: 60.175294, lon: 24.684855}  
    numItineraries: 3  
  ) {  
    itineraries {  
      legs {  
        startTime  
        endTime  
        mode  
        duration  
        realTime  
        distance  
        transitLeg  
      }  
    }  
  }  
}
```

Response with alternative itineraries (legs):

```
{  
  "data": {  
    "plan": {  
      "itineraries": [  
        {  
          "legs": [  
            {  
              "startTime": 1525451735000,  
              "endTime": 1525451820000,  
              "mode": "WALK",  
              "duration": 85,  
              "realTime": false,  
              "distance": 93.658,  
              "transitLeg": false  
            },  
            {  
              "startTime": 1525451820000,  
              "endTime": 1525452780000,  
              "mode": "SUBWAY",  
              "duration": 960,  
              "realTime": false,  
              "distance": 11398.66066011738,  
              "transitLeg": true  
            },  
            ...  
          ]  
        }  
      ]  
    }  
  }  
}
```

```

        "startTime": 1525452155000,
        "endTime": 1525452240000,
        "mode": "WALK",
        "duration": 85,
        "realTime": false,
        "distance": 93.658,
        "transitLeg": false
    },
    {
        "startTime": 1525452240000,
        "endTime": 1525453380000,
        "mode": "SUBWAY",
        "duration": 1140,
        "realTime": false,
        "distance": 13174.891143900431,
        "transitLeg": true
    },
    ...
    ]
}
}
}
}
}

```

Further examples are available at:

<https://digitransit.fi/en/developers/apis/1-routing-api/>

Whereas the Digitransit Routing API is at:

<https://api.digitransit.fi/routing/v1/routers/hsl/index/graphql>

Generic OTP Routing API is based on the more traditional REST API. The REST Resources available for the OTP API are the following:

- AlertPatcher
- BikeRental
- ExternalGeocoderResource
- GeocoderResource
- GraphInspectorTileResource
- IndexAPI
- LegendResource
- LIsochrome
- MuncoderServer
- PlannerResource
- PointSetResource
- ProfileResource
- RepeatedRaptorTestResource
- Routers
- SIsochrome
- ScenarioResource
- ScriptResource
- ServerInfo

- SimpleIsochrone
- Isochrone
- SurfaceResource
- TileService
- TimeGridWs
- UpdaterStatusResource

Of these, the *PlannerResource* is the primary entry point for the trip planning web service. All parameters are passed in the query string. These parameters are defined as fields in the abstract *RoutingResource* superclass, which also has methods for building routing requests from query parameters. This allows multiple web services to have the same set of query parameters. In order for inheritance to work, the REST resources are request-scoped (constructed at each request) rather than singleton-scoped (a single instance existing for the lifetime of the OTP server). The following resource is applicable:

`/routers/{routerid}/plan .`

The full list of GET parameters is available in the Table 10.

Name	Description
routerId	The routerId selects between several graphs on the same server. The routerId is pulled from the path, not the query parameters. Default to be used is <i>default</i> .
arriveBy	Whether the trip should depart or arrive at the specified date and time.
bannedAgencies	The comma-separated list of banned agencies.
bannedRoutes	The comma-separated list of banned routes. The format is agency_[routename][_routeid], so TriMet_100 (100 is route short name) or Trimet_42 (two underscores, 42 is the route internal ID).
bannedStops	A comma-separated list of banned stops. A stop is banned by ignoring its pre-board and pre-alight edges. This means the stop will be reachable via the street network. Also, it is still possible to travel through the stop. Just boarding and alighting is prohibited. The format is agencyId_stopId, so: TriMet_2107
bannedStopsHard	A comma-separated list of banned stops. A stop is banned by ignoring its pre-board and pre-alight edges. This means the stop will be reachable via the street network. It is not possible to travel through the stop. For example, this parameter can be used when a train station is destroyed, such that no trains can drive through the station anymore. The format is agencyId_stopId, so: TriMet_2107
bannedTrips	The comma-separated list of banned trips. The format is agency_trip[:stop*], so: TriMet_24601 or TriMet_24601:0:1:2:17:18:19
batch	If true, goal direction is turned off and a full path tree is built (specify only once)
bikeBoardCost	Prevents unnecessary transfers by adding a cost for boarding a vehicle. This is the cost that is used when boarding while cycling. This is usually higher than walkBoardCost.
bikeSpeed	The user's biking speed in meters/second. Defaults to approximately 11 MPH, or 9.5 for bikeshare.
bikeSwitchCost	The cost of the user fetching their bike and parking it again. Defaults to 0.
bikeSwitchTime	The time it takes the user to fetch their bike and park it again in seconds. Defaults to 0.
boardSlack	N/A
clampInitialWait	When subtracting initial wait time, do not subtract more than this value, to prevent overly optimistic trips. Reasoning is that it is reasonable to delay a trip start 15 minutes to make a better trip, but that it is not reasonable to delay a trip start 15 hours; if that is to be done, the time needs to be included in the trip time. This number depends on the transit system; for transit systems where trips are planned around the vehicles, this number can be much higher. For instance, it's perfectly reasonable to delay one's trip 12 hours if one is taking a cross-country Amtrak train from Emeryville to Chicago. Has no effect in stock OTP, only in Analyst. A value of 0 means that initial wait time will not be

	subtracted out (will be clamped to 0). A value of -1 (the default) means that clamping is disabled, so any amount of initial wait time will be subtracted out.
date	The date that the trip should depart (or arrive, for requests where arriveBy is true).
disableRemainingWeightHeuristic	If true, the remaining weight heuristic is disabled. Currently only implemented for the long distance path service.
fromPlace	The start location -- either latitude, longitude pair in degrees or a Vertex label. For example, 40.714476,-74.005966 or mtanycsubway_A27_S.
ignoreRealtimeUpdates	If true, realtime updates are ignored during this search.
intermediatePlaces	An ordered list of intermediate locations to be visited (see the fromPlace for format). Parameter can be specified multiple times.
locale	
maxPreTransitTime	The maximum time (in seconds) of pre-transit travel when using drive-to-transit (park and ride or kiss and ride). Defaults to unlimited.
maxTransfers	The maximum number of transfers (that is, one plus the maximum number of boardings) that a trip will be allowed. Larger values will slow performance, but could give better routes. This is limited on the server side by the MAX_TRANSFERS value in org.opentripplanner.api.ws.Planner.
maxWalkDistance	The maximum distance (in meters) the user is willing to walk. Defaults to unlimited.
minTransferTime	The minimum time, in seconds, between successive trips on different vehicles. This is designed to allow for imperfect schedule adherence. This is a minimum; transfers over longer distances might use a longer time.
mode	The set of modes that a user is willing to use, with qualifiers stating whether vehicles should be parked, rented, etc.
nonpreferredTransferPenalty	An additional penalty added to boardings after the first when the transfer is not preferred. Preferred transfers also include timed transfers. The value is in OTP's internal weight units, which are roughly equivalent to seconds. Set this to a high value to discourage transfers that are not preferred. Of course, transfers that save significant time or walking will still be taken. When no preferred or timed transfer is defined, this value is ignored.
numItineraries	The maximum number of possible itineraries to return.
optimize	The set of characteristics that the user wants to optimize for. "FLAT" or "GREENWAYS" or "QUICK" or "SAFE" or "TRANSFERS" or "TRIANGLE"
otherThanPreferredRoutesPenalty	Penalty added for using every route that is not preferred if user set any route as preferred, i.e. number of seconds that we are willing to wait for preferred route.
preferredAgencies	The comma-separated list of preferred agencies.
preferredRoutes	The list of preferred routes. The format is agency_[routename]_[routeid], so TriMet_100 (100 is route short name) or Trimet__42 (two underscores, 42 is the route internal ID).
reverseOptimizeOnTheFly	If true, this trip will be reverse-optimized on the fly. Otherwise, reverse-optimization will occur once a trip has been chosen (in Analyst, it will not be done at all).
startTransitStopId	A transit stop required to be the first stop in the search (AgencyId_StopId)
startTransitTripId	A transit trip acting as a starting "state" for depart-onboard routing (AgencyId_TripId)
time	The time that the trip should depart (or arrive, for requests where arriveBy is true).
toPlace	The end location (see fromPlace for format).
transferPenalty	An additional penalty added to boardings after the first. The value is in OTP's internal weight units, which are roughly equivalent to seconds. Set this to a high value to discourage transfers. Of course, transfers that save significant time or walking will still be taken.
triangleSafetyFactor	For bike triangle routing, how much safety matters (range 0-1).
triangleSlopeFactor	For bike triangle routing, how much slope matters (range 0-1).
triangleTimeFactor	For bike triangle routing, how much time matters (range 0-1).
unpreferredAgencies	The comma-separated list of unpreferred agencies.
unpreferredRoutes	The list of unpreferred routes. The format is agency_[routename]_[routeid], so TriMet_100 (100 is route short name) or Trimet__42 (two underscores, 42 is the route internal ID).

waitAtBeginningFactor	How much less bad is waiting at the beginning of the trip (replaces waitReluctance).
waitReluctance	How much worse is waiting for a transit vehicle than being on a transit vehicle, as a multiplier. The default value treats wait and on-vehicle time as the same. It may be tempting to set this higher than walkReluctance (as studies often find this kind of preferences among riders) but the planner will take this literally and walk down a transit line to avoid waiting at a stop. This used to be set less than 1 (0.95) which would make waiting offboard preferable to waiting onboard in an interlined trip. That is also undesirable. If we only tried the shortest possible transfer at each stop to neighboring stop patterns, this problem could disappear.
walkBoardCost	Prevents unnecessary transfers by adding a cost for boarding a vehicle. This is the cost that is used when boarding while walking.
walkReluctance	A multiplier for how bad walking is, compared to being in transit for equal lengths of time. Defaults to 2. Empirically, values between 10 and 20 seem to correspond well to the concept of not wanting to walk too much without asking for totally ridiculous itineraries, but this observation should in no way be taken as scientific or definitive. Your mileage may vary.
walkSpeed	The user's walking speed in meters/second. Defaults to approximately 3 MPH.
wheelchair	Whether the trip must be wheelchair accessible.

Table 10 Complete list of GET parameters for the Routing Baseline Services API

Example: Plan an itinerary from 60.163846, 24.909665 to 60.163696, 24.949265:

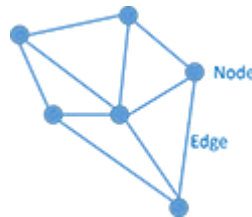
<http://localhost:8080/otp/routers/default/plan?fromPlace=60.163846,24.909665&toPlace=60.163696,24.949265&time=6.21pm&date=05-28-2018&mode=TRANSIT,WALK&maxWalkDistance=804&arriveBy=false&wheelchair=false&locale=en>

Response:

```
{
  "requestParameters": {
    "date": "05-28-2018",
    "mode": "TRANSIT,WALK",
    "arriveBy": "false",
    "wheelchair": "false",
    "fromPlace": "60.16384578852122,24.909696578979492",
    "toPlace": "60.16369632905707,24.949264526367184",
    "time": "6:21pm"
  },
  "legs": [
    {
      "startTime": 1527521314000,
      "endTime": 1527521639000,
      "departureDelay": 0,
      "arrivalDelay": 0,
      "realTime": false,
      "distance": 352.214,
      "pathway": false,
      "mode": "WALK",
      "route": "",
      "agencyTimeZoneOffset": 10800000,
      "interlineWithPreviousLeg": false,
      "from": {
        "name": "Origin",
        "lon": 24.909696578979492,
        "lat": 60.16384578852122,
        "departure": 1527521314000,
        "orig": "",
        "vertexType": "NORMAL"
      },
      "to": {
        "name": "Ruoholahti (M)",
        "stopId": "1:1201136",
        "stopCode": "0213",
        "lon": 24.91484,
        "lat": 60.16335,
        "arrival": 1527521639000,
        "departure": 1527521640000,
        "zoneId": "01",
        "stopIndex": 2,
        "stopSequence": 3,
        "vertexType": "TRANSIT"
      },
      "legGeometry": {
        "points": "uvenJsd`wCAk@A[?_@ASHA?EAs@JAAYn@CC_BPAAa@?KAs@?Q?IAu@AK?YFAF?A[?@_@AOD?BABAFAgSE?GE}C",
        "length": 33,
        "rentedBike": false,
        "transitLeg": false,
        "duration": 325.0,
        "steps": [
          {
            "distance": 34.709,
            "relativeDirection": "DEPART",
            "streetName": "service road",
            "absoluteDirection": "EAST",
            "stayOn": false,
            "area": false,
            "bogusName": false,
            "lon": 24.909703783685433,
            "lat": 60.16379871221838,
            "elevation": [ ]
          },
          {
            "distance": 87.806,
            "relativeDirection": "RIGHT",
            "streetName": "path",
            "absoluteDirection": "SOUTH",
            "stayOn": false,
            "area": false,
            "bogusName": true,
            "lon": 24.910329700000002,
            "lat": 60.1638201,
            "elevation": [ ]
          },
          {
            "distance": 69.821,
            "relativeDirection": "RIGHT",
            "streetName": "open area",
            "absoluteDirection": "SOUTH",
            "stayOn": true,
            "area": true,
            "bogusName": true,
            "lon": 24.911265800000002,
            "lat": 60.1635112,
            "elevation": [ ]
          },
          {
            "distance": 42.534000000000006,
            "relativeDirection": "RIGHT",
            "streetName": "steps",
            "absoluteDirection": "SOUTH",
            "stayOn": true,
            "area": false,
            "bogusName": true,
            "lon": 24.9123664,
            "lat": 60.1634645,
            "elevation": [ ]
          },
          {
            "distance": 117.344000000000002,
            "relativeDirection": "RIGHT",
            "streetName": "service road",
            "absoluteDirection": "SOUTH",
            "stayOn": true,
            "area": false,
            "bogusName": true,
            "lon": 24.9129128,
            "lat": 60.1634029,
            "elevation": [ ]
          }
        ]
      }
    }
  ],
  ...
}
```

2.2.3.1 Information Model

OTP performs routing on a graph structure, created from street data. Such street data needs individual streets to be connected in the data model. A correctly created OpenStreetMap⁶ data supports such linking. A graph consists of *Nodes* (*vertices* or *points*), connected to other Nodes by *Edges* (*arcs* or *lines*). In path planning, a *weight* between two Nodes determines the cost of traveling the connecting Edge. For example, in the case of routing cars in a city, the weight might represent the expected time of travel, i.e. distance divided by speed limit. A routing system that supports real time updates might update such weights according to an estimated traffic situation, i.e. assigning higher weights during a traffic hour.



Digitransit OTP Service requests are based on the GraphQL, allowing queries for itinerary *plans*, disruption *alerts*, public transport *routes* and public transport *stops*.

Generic OTP Service Data Elements are:

- Agency
- AgencyAndId
- AgencyList
- AlertPatch
- AlertPatchCreationResponse
- alertPatchResponse
- AlertPatchSet
- BikeRentalStationList
- Calendar
- CalendarData
- CalendarDate
- debugOutput
- geocoderResults
- graphMetadata
- Headsign
- InspectorLayersList
- modeList
- MyMap
- response
- route
- RouteData
- RouteDataList
- RouteList
- RouterInfo
- RouterList
- RouteVariant
- serverInfo

⁶ <https://www.openstreetmap.org>

-
- Stop
 - StopList
 - StopTimeList
 - transitError
 - trip
 - trip
 - TripList
 - tripMatch

Generic OTP Service Data Types are:

- absoluteDirection
- agencyAndId
- agencyAndIdType
- agencyList
- agencyType
- alert
- alertPatch
- alertPatchCreationResponse
- alertPatchResponse
- alertPatchSet
- bikeRentalStation
- bikeRentalStationList
- calendarData
- debugOutput
- encodedPolylineBean
- entry
- fare
- fareType
- geocoderResult
- geocoderResults
- graphMetadata
- headsignsInfo
- inspectorLayer
- inspectorLayersList
- itinerary
- leg
- mapType
- mavenVersion
- message
- modeList
- money
- note
- place
- plannerError
- relativeDirection
- response
- routeData
- routeDataList
- routeList

-
- routerInfo
 - routerList
 - routeType
 - routeVariant
 - serverInfo
 - serviceCalendarDateType
 - serviceCalendarType
 - stopList
 - stopTime
 - stopTimeList
 - stopType
 - transitError
 - translatedString
 - TraverseMode
 - tripList
 - tripMatch
 - tripPlan
 - tripsModelInfo
 - tripType
 - walkStep
 - wrappedCurrency

For more details, the reader is invited to visit the full documentation on generic OTP Data Model with Data Elements and Data Types⁷.

2.2.3.2 Interaction diagram (with the external world)

A typical way for a user to make an itinerary request from OTP is via a map, where the user defines the start and end points, then specifying other request parameters in a dialog box. These parameters are then attached to the API call (see Section 2.2.3 API), handled within OTP (see Section 2.2.4.1 Operation Model for details), and response received and rendered. Figure 2 presents the generic OTP map interface, where a user has defined a route request in Helsinki using the generic OTP service, receiving an itinerary including walking and travel by metro.

⁷ <http://dev.opentripplanner.org/apidoc/1.0.0/index.html>

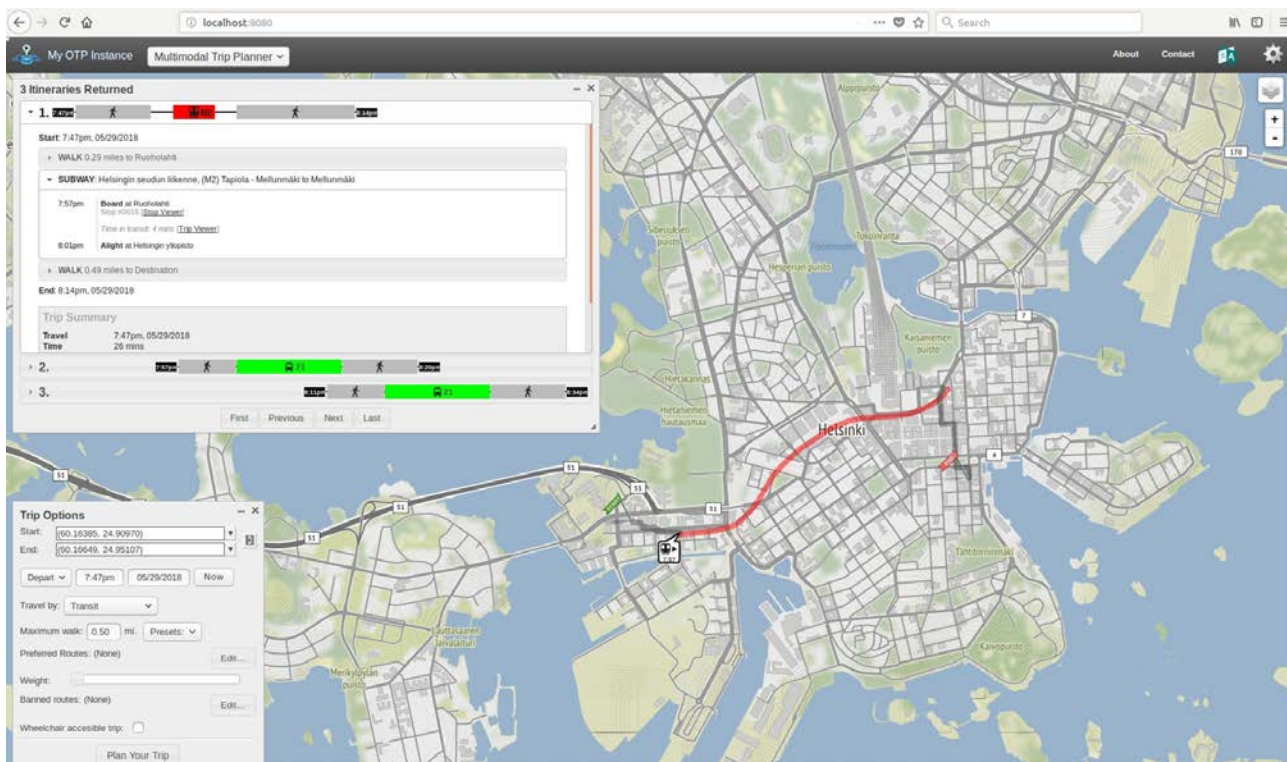


Figure 2. The web interface of the generic OTP service.

2.2.4 Service Architecture

OpenTripPlanner operates in two modes: as an instance running via Grizzly server (web server), which is suited for planning trips, and an analyst mode in which the complete network can be analyzed. Both of them refer to the same core: a routing Java library which is responsible for computing the multi modal transport trips. On top of this core, several services are built which call its internal functions.

Here are presented the two of them that are used on a daily basis. The first one is Routing API, a RESTful (REST is Representational State Transfer) web service which responds to journey planning requests. This, with an appropriate mapping library can form a friendly web GUI (Graphical User Interface) that provides basic planning features. The other one is Script API that introduces users the ability to create automation mechanisms and functions for OTP with the use of their favorite scripting language. At that moment only a limited number of these are supported.

The Analyst can be executed in two configurations: the basic Analyst and the Batch Analyst. The basic Analyst Web Services provide basic network analysis results, while the Batch Processor handles more complex examination tasks. It allows an implementation of very open-ended configuration, which includes, for example, information regarding terrain's population rates and therefore provides a powerful tool for visualizing how transport networks affect access to urban facilities. Both the Analyst and Batch Analyst do not make use of services, but rather directly call methods inside the core.

When it comes to the core itself, it incorporates two separate services for its proper functionality: OTP Transit Index API and Graph Builder. The Transit Index delivers information to the core,

recovering them from GTFS (General Transit Feed Specification)⁸ files, which specify routes, stops and frequencies of all transit lines. The Graph Builder composes graphs from available OpenStreetMap (OSM) databases, which handle open source geographical data. These are created mostly by community and available to anyone, who is willing to utilize them.

Additionally, the Graph Builder can be invoked by OTP instance via Graph Visualizer. This component provides a basic interface for recovering edges and vertices' metadata.

A visualization of the as-intended development view has been composed according to the prior description and depicted on the graph in Figure 3.

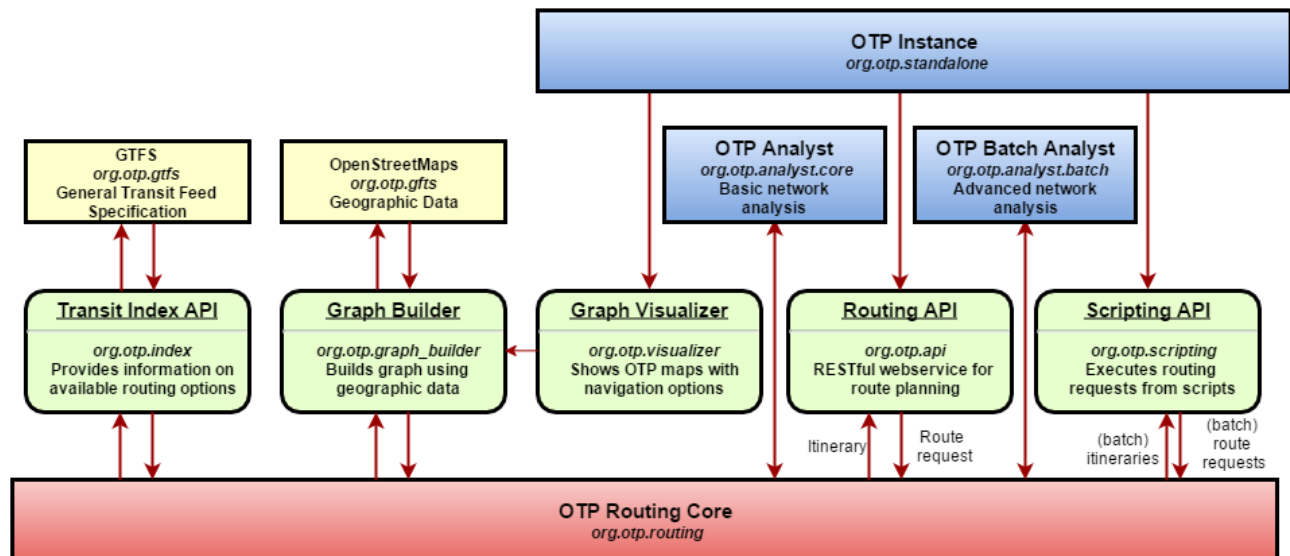


Figure 3. Development view of OTP.

2.2.4.1 Operation model (Message flows between components)

A user provides his requests to a Routing API by specifying the origin, destination and preferences for example at the web browser. This request is send via the RESTful web service to the OTP instance. Figure 4 presents this internal process. A routing request is handled by the Routing Core module. The core creates a separate Routing Context for each routing request. This context includes the user input and other parameters, as well as the graph and transit feeds. If requested, the real-time feeds for transit and historical data on traffic speeds is included in the Routing Context as well. The Context is then given to the PathFinder. The PathFinder is responsible for actually searching possible routes from origin to destination. This is done by applying the A star algorithm. Several heuristics are used in this algorithm to speed up, depending on the type of request. This leads to a ShortestPathTree with the discovered itineraries. From this list the itineraries are taken that actually arrive at the wanted destination, also called States. These are banned by the PathFinder from the considered paths for the next searches, to create more variety in the found routes. The PathFinder continues iterating until a certain time limit is reached or the number of paths found is sufficient. Next,

⁸ <https://developers.google.com/transit/gtfs/>

the found paths are returned to the user.

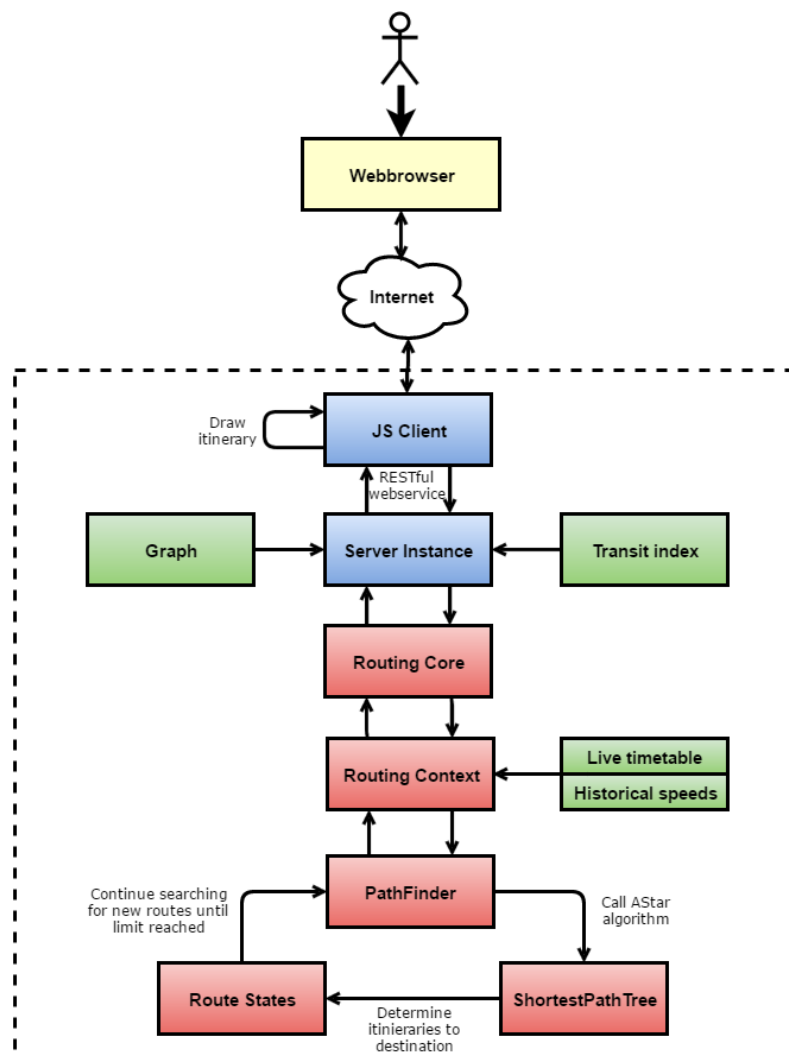


Figure 4. Functional view of OTP: how a request is handled.

2.2.4.2 Non-functional requirements

The service depends on OpenStreetMap data, and public transport schedule data in GTFS format. These need to be pre-fetched prior to launching OTP. A routing graph is created based on these inputs, and can be stored to disk or kept in memory only. The routing service is rather resource-heavy, so a lot of memory is needed for an instance, i.e. 2-8GB, depending on the size of the graph (map data) and to some extent the density of public transport service lines.

2.2.5 Service implementation

2.2.5.1 Service implementation IT environment (dependencies like SQL db)

The OpenTripPlanner depends on map, schedule and public transport route data. As a pre-requirement, local public transport schedule data needs to be processed into GTFS⁹ format. A copy of local OpenStreetMap (OSM) data is downloaded, and verified. Routing data is downloaded in GTFS format, verified, fit to the verified OSM data, filtered and assigned identifiers. Finally, a graph for the actual routing algorithm is build based on the OSM data and installed on local filesystem. The build process for Digitransit OTP data container is explained here: <https://github.com/HSLdevcom/OpenTripPlanner-data-container>. Starting the generic OTP server and simultaneously generating the graph to memory (without storing a data container to disk) at start time is presented in the Installation and Admin Guide.

The generic OpenTripPlanner is distributed as a single stand-alone runnable JAR file. These JARs are deployed to the Sonatype OSSRH Maven repository¹⁰, and release versions are synced to the Maven Central repository. Most people will want to go to the OTP directory at Maven Central¹¹, navigate to the directory for the highest version number, and download the file whose name ends with `.shaded.jar`.

You may also choose to build OTP from its source code. If you will be modifying OTP you will need to know how to rebuild it (though your IDE may take care of this build cycle for you). If you have the right software installed, building OTP locally from its source code is not particularly difficult. You should only need the following software:

- Git, a version control system
- Java Development Kit, preferably version 8 (AKA version 1.8)
- Maven, a build and dependency management system

The installation and admin guide provides a set of commands that will ensure that necessary components are available for executing the generic OTP.

2.2.5.2 Service deployment dependencies (e.g. Orion Broker, oAuth server etc.)

The service needs as primary data sources both OpenStreetMap and GTFS data, which need to be pre-fetched prior to launching the OTP service.

2.2.5.3 Installation and admin guide

These instructions assume that system is being installed on machine with Ubuntu 16.04 OS.

1. Install Docker

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

⁹ <https://developers.google.com/transit/gtfs/reference/>

¹⁰ <http://central.sonatype.org/pages/ossrh-guide.html>

¹¹ <https://repo1.maven.org/maven2/org/opentripplanner/otp/>

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce
```

2. Get GTFS data

The best option is to get the data directly from transit operator or agency but Transitland¹² and TransiFeeds¹³ services also provide lists of feeds.

If you are going to use data from Helsinki (HSL) use following command:

```
wget http://dev.hsl.fi/gtfs/hsl.zip
```

If your local public transport schedule data is not available in GTFS format, you need to do this formatting yourself. Please follow the guidelines by gtfs.org¹⁴.

3. Get OpenStreetMap data

You also need OSM data for the same area as the GTFS -data is from. You can download it from the Geofabrik's servers¹⁵.

Files provided by the Geofabrik are quite large, so you need to trim them further with osmconvert.

To do this, you need to figure out the bounding box you are interested in. This can be done for example with the bounding box tool¹⁶. The CSV (Comma Separated Values) option in that tool produces exactly the format expected by the osmconvert -b switch.

For example, this is how you would get data for Helsinki Region:

Download OSM-data for Finland:

```
wget http://download.geofabrik.de/europe/finland-latest.osm.pbf
```

Clip Helsinki region with osmconvert:

```
osmconvert finland-latest.osm.pbf -b=24.002,59.9774,25.7323,60.6951 --complete-ways -o=helsinki.pbf
```

4. Start OTP using a Docker image

Before you start the OTP, create new folder for the data. E.g. ~/otpdata and copy pbf and gtfs files into that. e.g.

```
mkdir ~/otpdata
cp helsinki.pbf hsl.zip ~/otpdata
```

¹² <https://transit.land/feed-registry>

¹³ <http://transitfeeds.com>

¹⁴ <http://gtfs.org/best-practices/>

¹⁵ <http://download.geofabrik.de>

¹⁶ <https://boundingbox.klokantech.com>

Then you simply run the OTP from the Docker image, which is automatically fetched from the Maven repository and installed to your local filesystem:

```
docker run \  
-e JAVA_MX=8G \  
-v ~/otpdata:/data:ro \  
-p 8080:8080 \  
opentransitmap/opentripplanner \  
run.sh --build /data --inMemory
```

The OTP should start and build the graph from given data. `--inMemory` declares that OTP will not store data into the disk. If you wish it to do so, you need to check the documentation of advanced usage.

OTP is quite memory intensive application, so you probably need to adjust the `JAVA_MX` - parameter according to your resources (size of the graph and the server)

After few moments (this can take over 20 minutes, depending on your graph size and resources. Be patient), the OTP should print "Grizzly server running.", after that the OTP server should be running in <http://localhost:8080>

2.3 Parking probability estimation

2.3.1 Description

Given a time slot in the near future (e.g. in 20 minutes from now) and a concrete area in the city (among a set of pre-defined by the city ones), this service will return an estimation of the parking probability, based on the current and historical occupancy of the selected area. This estimation can be also assisted by additional context information, as for example, weather forecast or traffic status.

This service will rely on ParkingSpot¹⁷ entities that provide updated status of each monitored parking lot. To define the parking areas, this service will use OnStreetParking¹⁸ entities that group different ParkingSpot entities. The way these ParkingSpots are linked may define additional functionalities of this service (e.g. estimation of parking probability for disabled drivers, electric cars, wide vehicles, caravans, etc.). OnStreetParking entities are referred to open parking lots in the city, but the service also can include OffStreetParking¹⁹ entities, to cover also for example private parking premises.

2.3.2 Application theme Functional Requirement addressed

This baseline service targets the FR-20 of the Multi-Modal transportation application theme related since it estimates the availability of parking spots in the near futures.

FR code	FR description	Comments
FR-20	The application may be able to estimate when and where it will be possible to find a free parking lot.	The parking probability estimator service has been conceived specifically for addressing this purpose.

Table 11 Multi-modal transportation application theme FRs addressed

2.3.3 Interface

2.3.3.1 Parking estimation API

This section provides a first description of the Parking Service API, focused on its main functionalities. The final implementation of the APIs will be provided later.

The Parking Service APIs will retrieve information about parking areas belonging to a city and their related parking spots. Moreover, the Parking Service APIs will provide the real time status of a parking area, and the expected one in a given time slot, both in terms of number and percentage of free/occupied parking lots.

The draft technical description of the Parking Service APIs is also online available²⁰

The first version of Parking Service APIs consists of three RESTful services, in particular:

¹⁷ <http://fiware-datamodels.readthedocs.io/en/latest/Parking/ParkingSpot/doc/spec/index.html>

¹⁸ <http://fiware-datamodels.readthedocs.io/en/latest/Parking/OnStreetParking/doc/spec/index.html>

¹⁹ <http://fiware-datamodels.readthedocs.io/en/latest/Parking/OffStreetParking/doc/spec/index.html>

²⁰ <https://synchronicityparkingservice.docs.apiary.io>

- List Parking Areas: this method will return the list of the registered Parking Areas and it is described in Table 12.
- Parking Area Detail: this method will return the detail and the current status of a Parking Area and it is described in Table 13.
- Free Parking Spots Estimation: this method will return an estimation of the free Parking Spots of a Parking Area in a specific time of the day and it is described in Table 14.

It is important to underline that the methods *List Parking Areas* and *Parking Area Detail* provide an easier and contextualized way to retrieve information from the context broker that manages the entities used by the service; indeed, it will be possible to retrieve the same information using directly the APIs of the context broker, for instance the one used by a RZ.

Furthermore, a short description of data structures adopted in the APIs is also reported in this section.

API Name		List Parking Areas			
Description		This service will return the paginated list of the registered Parking Areas (OnStreetParking or OffStreetParking).			
End-Point		/parkingareas{?rows,offset,type}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandato ry	Default	Description
rows	Query	number	No	20	This parameter is used to paginate the results and represents the first element of the result list where to start.
offset	Query	number	No	0	This parameter is used to paginate the results and represents the first element of the result list where to start.
type	Query	Enum (OnStreetParking, OffStreetParking)	No	-	This parameter, if present, is used to filter the response returning only the type provided (OnStreetParking or OffStreetParking)
Responses					
HTTP Status Code		Content-type	Entity		Description
200		application/json	Array of OnStreetParking/OffStreetParking		The list of the Parking Areas.

400	application/json	ErrorResponse	Bad Request (e.g. rows parameter not numerical)
500	application/json	ErrorResponse	Server error

Table 12 List Parking Areas API

API Name		Parking Area Detail			
Description		This service will return the detail of the specific Parking Area (OnStreetParking or OffStreetParking) identified by the parameter parkingareaID.			
End-Point		/parkingareas/{parkingareaID}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
parkingareaID	Path	String	Yes	-	The unique identifier of the parking area.
Responses					
HTTP Code	Status	Content-type		Entity	Description
200		application/json		OnStreetParking/OffStreetParking	The requested Parking Area
404		application/json		ErrorResponse	Parking Area Not Found
500		application/json		ErrorResponse	Server error

Table 13 Parking Area Detail API

API Name	Free Parking Spots Estimation
Description	This service will return the estimation of the free spots available in the Parking Area specified by the parameter parkingareaID. The query parameter expectedArrivalTime is mandatory. The query parameter category is optional.

End-Point		/parkingareas/{parkingareaID}/freespotsEstimation{?expectedArrivalTime, category}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
parkingareaID	Path	String	Yes	-	The unique identifier of the parking area.
expectedArrivalTime	Query	Enum (10 min, 20 min, 30 min, 40 min, 50 min, 1h)	Yes	-	This parameter provides the expected arrival time.
category	Query	Enum (none, disable, electric, wide vehicles, caravans)	No	none	This parameter, if provided, is used to filter the Parking Spots using a dedicated category, the default value is none.
Responses					
HTTP Code	Status	Content-type	Entity		Description
200		application/json	OnStreetParking/OffStreetParking		The selected parkingareaID with a new attribute that includes the estimation of the probability of finding a free parking lot
400		application/json	ErrorResponse		Bad Request (e.g. wrong expectedArrivalTime value)
404		application/json	ErrorResponse		Parking Area Not Found
500		application/json	ErrorResponse		Server error

Table 14 Free Parking Spot Estimation API

2.3.3.1.1 Data Structures

This section provides the description of the entities used in the Parking Service APIs.

The entities reported in this section are:

- **ErrorResponse**: this entity provides error messages that may occur during the execution of a process; its description is reported in Table 15.
- **OnStreetParking/OffStreetParking**: these entities model a parking area that could be place alongside a street (**OnStreetParking**) or not (**OffStreetParking**); their descriptions are reported in Table 16.

Entity Name	ErrorResponse	
Description	This entity is used to provide information about errors occurred during the execution of a request through the Parking Service APIs. Error codes and messages will be defined during the project. Specific error codes and relative messages will be defined later.	
Attributes		
Name	Type	Description
description	String	A string with the additional description of the error
error	String	A textual description of the error.

Table 15 ErrorResponse Entity

Entity Name	OnStreetParking/OffStreetParking	
Description	These entities represent generic parking areas that could be placed alongside a street or not. In this table their main attributes are reported. For more details, please refer to OnStreetParking ²¹ and OffStreetParking ²² FIWARE data models.	
Attributes		
Name	Type	Description
id	String	A string that represent the unique identifier of the Parking Area.

²¹ <http://fiware-datamodels.readthedocs.io/en/latest/Parking/OnStreetParking/doc/spec/index.html>

²² <http://fiware-datamodels.readthedocs.io/en/latest/Parking/OffStreetParking/doc/spec/index.html>

type	String	Entity type. It must be equal to OnStreetParking or OffStreetParking.
name	String	The name of the Parking Area
description	String	The description of the Parking Area.
category	Array of String	Identify the category of the parking area (e.g.: for disabled, for residents, etc.)
location	Object	Location of the observed traffic flow represented by a GeoJSON.
address	String	Registered onstreet/offstreet parking area civic address
totalSpotNumber	Number	The total number of parking spots offered by the parking area.
availableSpotNumber	Number	The number of parking spots available.
extraSpotNumber	Number	Extra spots reserved for special purposes and usually require a permit
refParkingSpot	Array of ParkingSpot	List of references to ParkingSpot

Table 16 OnStreetParking/OffStreetParkingParkingArea Entities

2.3.3.2 Information Model

This section provides a high level description of the entities used by the service to estimate parking spots allowance as referred in section 1.1.1 and the relations among them. The descriptions will take into account only those attributes effectively useful for the estimation algorithm, thus omitting those ones that are not relevant.

The single parking spot is described by an instance of the ParkingSpot entity as defined by the FIWARE NGSI data-model²³. As reported in the FIWARE documentation: *a parking spot is an area well delimited where one vehicle can be parked. The aim of this entity type is to monitor the status of parking spots individually. An entity of type ParkingSpot cannot exist without a containing entity of type OnStreetParking or OffStreetParking.*

Table 17 reports a description of the relevant attributes used by the estimation parking allowance service.

²³<http://fiware-datamodels.readthedocs.io/en/latest/Parking/ParkingSpot/doc/spec/index.html>

Parking Spot		
Attribute	Value	Notes
id	Entity's unique identifier	
type	Entity type	It must be equal to ParkingSpot
name	Name of the parking spot.	It can denote the number or label used to identify it within a parking site
description	Text	Description about the parking spot
width	Numeric	Width of the parking spot
length	Numeric	Length of the parking spot
location	Geolocation of the parking spot, represented by a GeoJSON Point	Not nullable (if address is not defined).
address	Registered parking spot civic address	Not nullable (if location is not defined).
status	text: ("occupied", "free", "closed", "unknown")	Status of the parking spot from the point of view of occupancy
	timestamp: datetime	(optional) Timestamp which reflects the date when the attribute value was obtained
refParkingSite	Parking site ID to which the the parking spot belongs to	Reference to an entity of type OnStreetParking or type OffStreetParking, depending on the value of the category attribute
category	text: ("onstreet", "offstreet")	Category of the parking spot
otherAttributes		other attributes to denote the entity not relevant for the estimation algorithm

Table 17 ParkingSpot Entity

An area that bundles a set of single parking spots that is accessible directly from a road is described by an instance of the OnStreetParking entity as defined by the FIWARE NGSI data-model²⁴. Table 18 reports a description of the relevant attributes used by the estimation parking allowance service.

On Street Parking		
Attribute	Value	Notes
id	Entity's unique identifier	
type	Entity type	It must be equal to "OnStreetParking"
name	Name of the onstreet parking area.	It can denote the number or label used to identify it within a parking site
description	text	Description about the onstreet parking area
category	list of text:	Street parking categories (eg. "forDisabled", "forResidents" etc)

²⁴<http://fiware-datamodels.readthedocs.io/en/latest/Parking/OnStreetParking/doc/spec/index.html>

location	Geolocation of the onstreet parking area, represented by a GeoJSON multipolygon	Not nullable (if address is not defined).
address	Registered onstreet parking area civic address	Not nullable (if location is not defined).
totalSpotNumber	numeric	The total number of spots offered by this parking site
availableSpotNumber	numeric	The number of spots globally available
	<i>timestamp: datetime</i>	(optional) Timestamp which reflects the date when the attribute value was obtained
extraSpotNumber	numeric	Extra spots are those reserved for special purposes and usually require a permit
	<i>timestamp: datetime</i>	(optional) Timestamp which reflects the date when the attribute value was obtained
refParkingSpot	List of references to ParkingSpot entities ID	(optional)
otherAttributes		other attributes to denote the entity not relevant for the estimation algorithm

Table 18 OnStreetParking Entity

An area that bundles a set of single parking spots that is not directly accessible from a road and is managed independently and with suitable and clearly marked access points (entrances and exits) is described by an instance of the OffStreetParking entity as defined by the FIWARE NGSI data model²⁵.

Table 19 reports a description of the relevant attributes used by the estimation parking allowance service.

Off Street Parking		
Attribute	Value	Notes
id	Entity's unique identifier	
type	Entity type	It must be equal to " OffStreetParking "
name	Name of the onstreet parking area.	It can denote the number or label used to identify it within a parking site
description	text	Description about the onstreet parking area
category	list of text:	Offstreet parking categories (eg. " <i>public</i> ", " <i>private</i> ", etc)
location	Geolocation of the offstreet parking area, represented by a GeoJSON multipolygon	Not nullable (if address is not defined).

²⁵ <http://fiware-datamodels.readthedocs.io/en/latest/Parking/OffStreetParking/doc/spec/index.html>

address	Registered offstreet parking area civic address	Not nullable (if location is not defined).
status	list of text: ("open", "closed", "full", etc)	(optional) Status of the offstreet parking area
	<i>timestamp: datetime</i>	(optional) Timestamp which reflects the date when the attribute value was obtained
totalSpotNumber	numeric	The total number of spots offered by this parking site
availableSpotNumber	numeric	The number of spots globally available
	<i>timestamp: datetime</i>	(optional) Timestamp which reflects the date when the attribute value was obtained
extraSpotNumber	numeric	Extra spots are those reserved for special purposes and usually require a permit
	<i>timestamp: datetime</i>	(optional) Timestamp which reflects the date when the attribute value was obtained
refParkingSpot	List of references to ParkingSpot entities ID	(optional)
otherAttributes		other attributes to denote the entity not relevant for the estimation algorithm, such as fee policy, contact info, facilities etc

Table 19 OnStreetParking Entity

Figure 1 illustrates the relations between ParkingSpot, OnStreetParking and OffStreetParking entities.



Figure 5 ParkingSpot, OnStreetParking and OffStreetParking relations

2.3.3.3 Interaction diagram (with the external world)

This section describes the high-level interactions between a generic Client Application and the Synchronicity's Parking Backend Services, taking advantage of the draft version of the Parking APIs defined in Section 2.2.3.

List Parking Areas

This API provides a list of the Parking Areas registered in the platform. This list is paginated using the rows and offset parameter. The type parameter, if provided, is used to filter the results between OnStreetParking and OffStreetParking entities. The expected interaction is depicted in Figure 6.

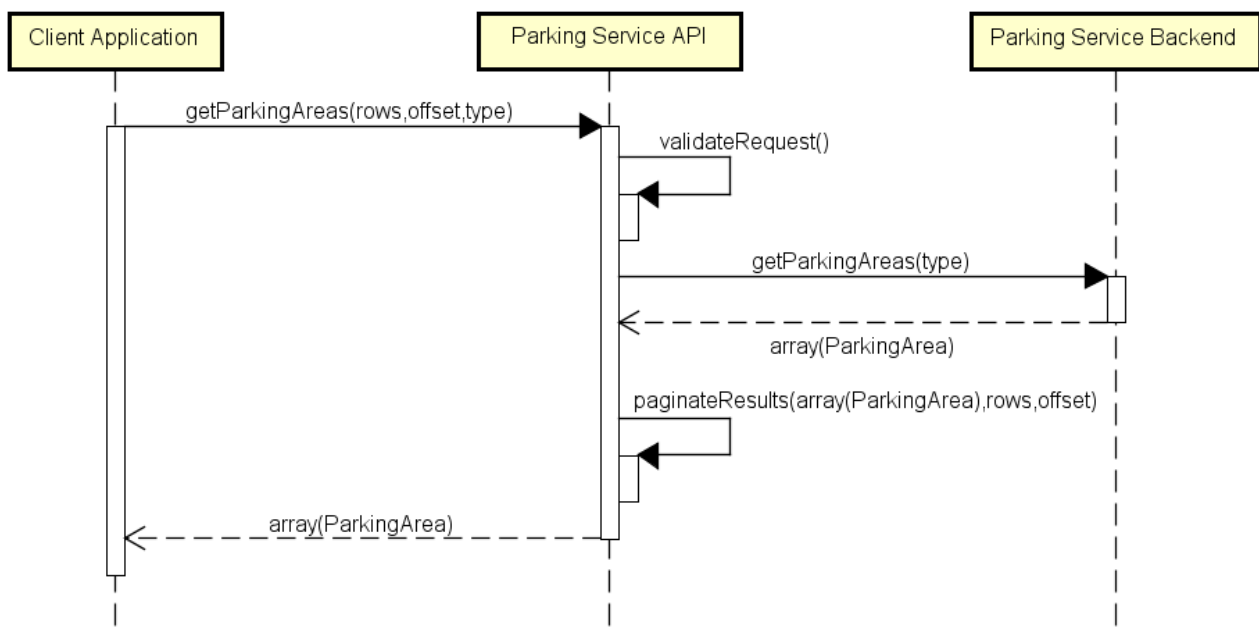


Figure 6 List Parking Areas

The Client Application requests for the registered Parking Areas to the Parking Service API, this component will validate the request and forward it to the Parking Service Backend who will retrieve the registered Parking Areas. The resulting list will be paginated and returned to the Client Application.

Parking Area Detail

This API will return the detail of a single Parking Area identified by the parkingareaID parameter. Figure 7 illustrates the interactions among components.

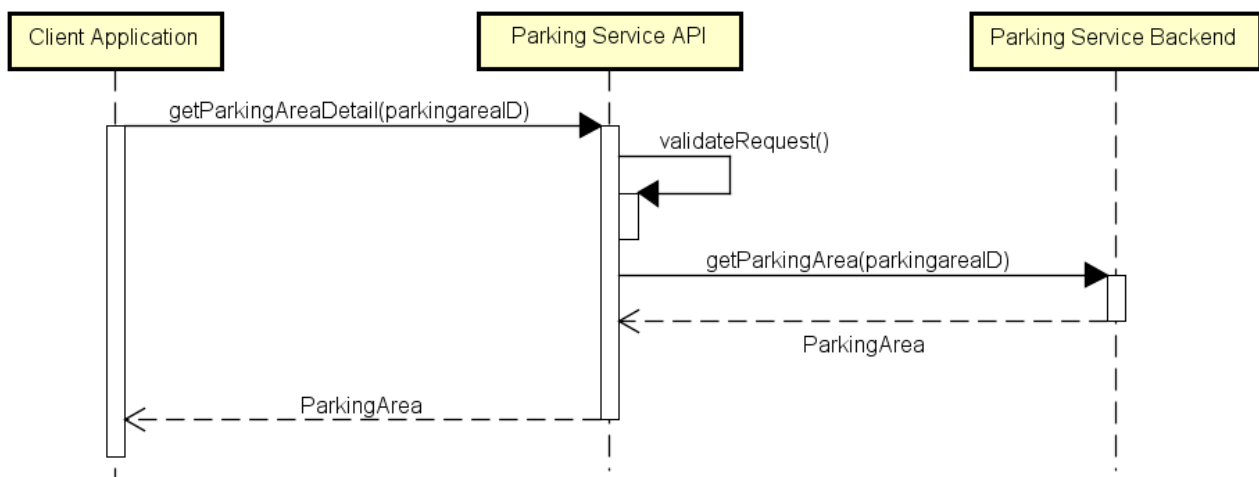


Figure 7 Parking Area Detail

The Client Application request for the details of a Parking Area registered in the platform. The Parking

Service API will validate the request and forward it to the Parking Service Backend who will return the requested Parking Area to the Parking Service API. This component will then forward the response to the Client Application.

Free Parking Spots Estimation

This API will use both current and historical Parking Area data in order to estimate the probability of finding a free parking lot in a specified time of the day provided for a single Parking Area. The sequence diagram of the underlying operations is showed in Figure 8.

The Client Application requests for a free spots estimation of a Parking Area identified by the parameter `parkingareaID`, the parameter `expectedArrivalTime` represents the expected day and time to search for free spots and the category (disable, pregnant, other) parameter, if provided, is used to filter the Parking Spots. The Parking Service API will validate the request and will forward it to the Parking Service Backend. This component will retrieve the historical data of the requested Parking Area and will use these values and the requested `expectedArrivalTime` to estimate the number of free spots of the Parking Area. This value will be provided to the Client Application through the Parking Service API.

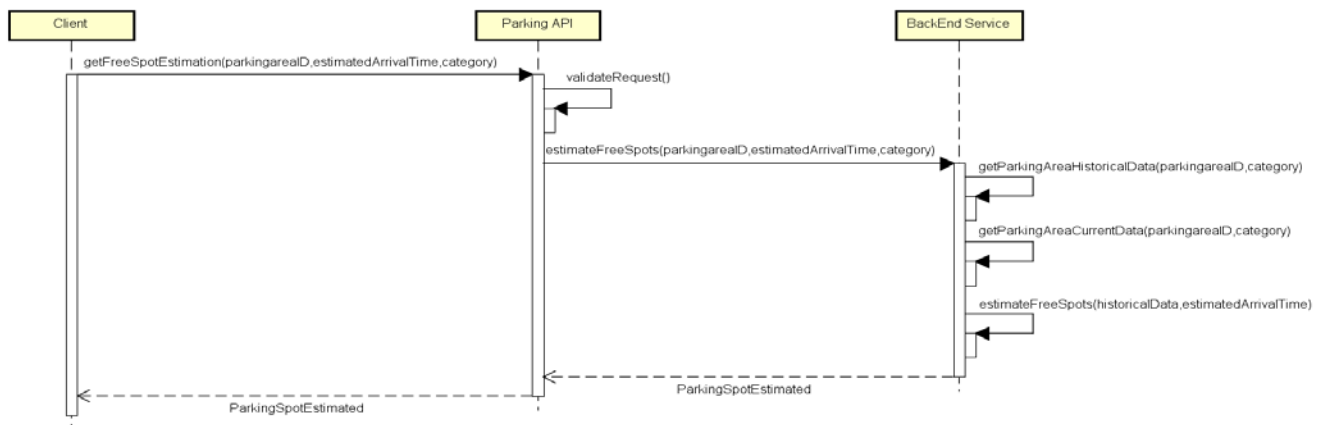


Figure 8 Free Parking Spots Estimation

2.3.4 Service Architecture

This section will go through the diverse components that compose the service, as well as will provide a description of the operational model to follow, listing the diverse messages that may flow among the service components. It will also address the series of non-functional requirements that is convenient to cope with.

2.3.4.1 Service components

As depicted in Figure 9, the service relies on SynchroniCity architecture and defined information and data models (SynchroniCity, D2.2, 2018) to capture the required information. It will also provide a RESTful standard based API to request the outcomes.

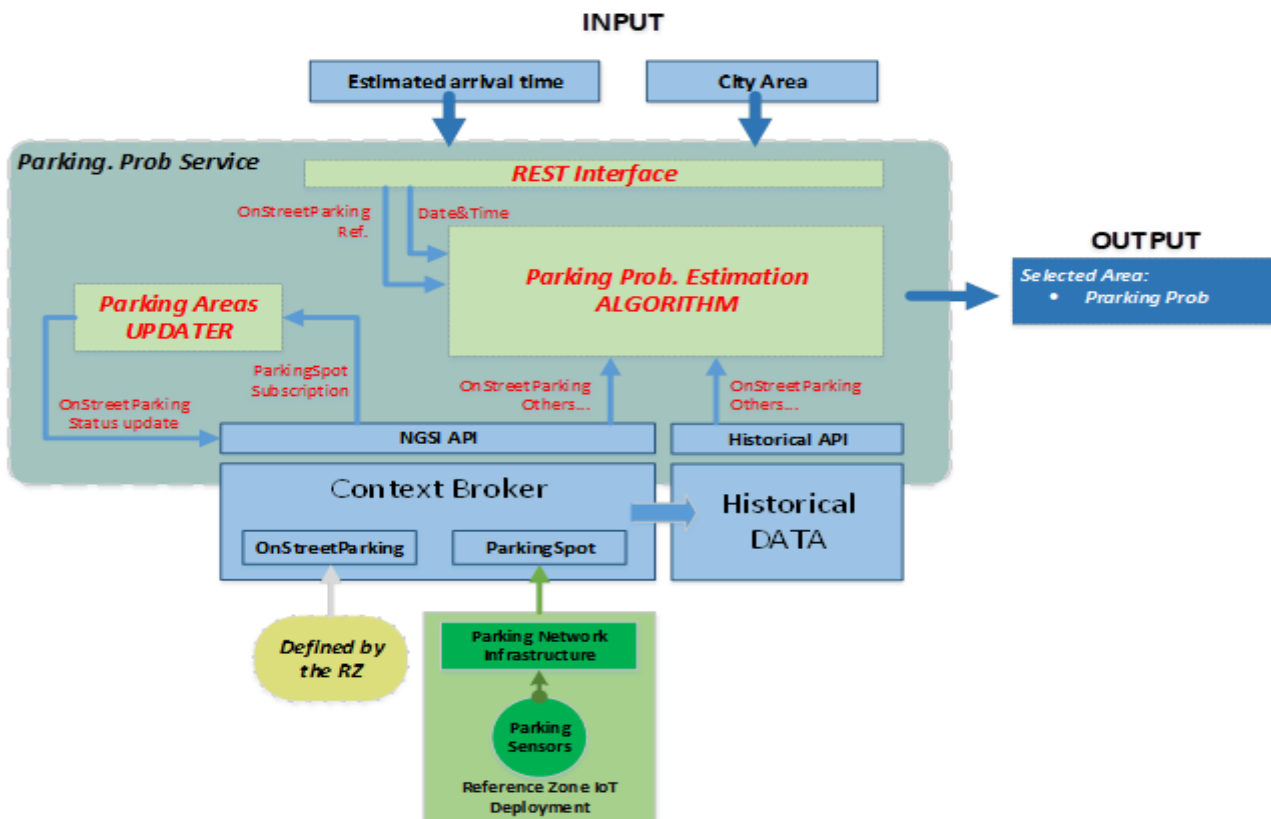


Figure 9 Parking probability estimator service architecture and interaction with the SynchroniCity framework

- Data Sources:** this block represents the data sources required to implement the service's functionalities. The initial implementation of the service is based on current status (context information) and historical evolution (historical data) of the sensed parking lots and their grouping on city areas. This is mapped according the SynchroniCity datamodels:
 - ParkingSpot²⁶:** a parking spot is an area well delimited where one vehicle can be parked. The aim of this entity type is to monitor the status of parking spots individually. It is a requirement, for the service, that the involved city/RZ has a parking sensors/monitoring system that provides this info.
 - OnStreetParking²⁷:** defines a site, open space zone, on street, (metered or not) with direct access from a road, intended to park vehicles, that groups a set of defined/identified ParkingSpots entities. These parking lots grouping should be defined by the city/RZ. The way this is done may add extra features to the service, as for instance the availability of parking lots for disabled people or last mile vehicles.

Depending on other available information sources and/or IoT deployments on each RZ, the basic functionalities could be enriched with:

²⁶ <http://fiware-datamodels.readthedocs.io/en/latest/Parking/ParkingSpot/doc/spec/index.html>

²⁷ <http://fiware-datamodels.readthedocs.io/en/latest/Parking/OnStreetParking/doc/spec/index.html>

- o WeatherForecast²⁸ and/or WeatherObserved²⁹ entities that add current and historical data about weather status which may influence on the parking usage.
- o OffStreetParking³⁰ that describes a site, off street, intended to park vehicles, managed independently and with suitable and clearly marked access points (entrances and exits). This may refer to private indoor parking areas in the city.

All these data sources will be mapped into Context Information entities, according NGSI data model, and managed through the SynchroniCity Context Broker component.

- **Parking Areas Updater:** this part of the service updates each OnStreetParking entity every time an associated ParkingSlot entity changes its status. This way, all the OnStreetParking context entity will inform about how many free parking lots are currently available within its corresponding city area. This code is based on NGSI subscriptions capabilities and its first implementation relies on FIWARE NGSIv2 API subscription method. It internally stores the relationships that defines each OnStreetParking entity (the set of ParkingSpots entities included on each one) and creates a HTTP listener that receives the NGSI notifications with the information related to the changing ParkingSpot entities.
- **Parking Probability Estimation Algorithm:** this is the core of the service. It collects info from the SynchroniCity platform (Context Broker and Data Storage - Historical data -) to provide an estimated value representing the parking probability for a specified area within the indicated time slot. The first version of this algorithm will work according to pre-defined OnStreetParking entities (each OnStreetParking entity will represent an area within the city) and the estimated parking probability will be given for these entities. In the same way, this first version will consider only current status and historical data related to these entities. Further versions will also include in the algorithm info related to weather status and/or traffic. This component has been thought to be easily replaced by more elaborated algorithms that may also include other relevant information sources or processes (adaptive algorithms, machine learning, etc.) that provides a more accurate estimation.
- **SynchroniCity Framework components** required to implement the basic service: the Parking Estimation Service will require form:
 - o SynchroniCity Context Manager component: through NGSI interface to provide current status (and location) of OnStreetParking entities. First service version will use the NGSIv2 API implementation provided by the FIWARE Orion Context Broker.
 - o SynchroniCity Historical Data component: through Sync. Historical Data API to provide OnStreetParking entities historical data to analyse. First version will be developed relying on the Historical Data Access API proposed by SynchroniCity. Final version will be compliant with ETSI NGSI-LD standard specifications.

2.3.4.2 Operation model (Message flows between components)

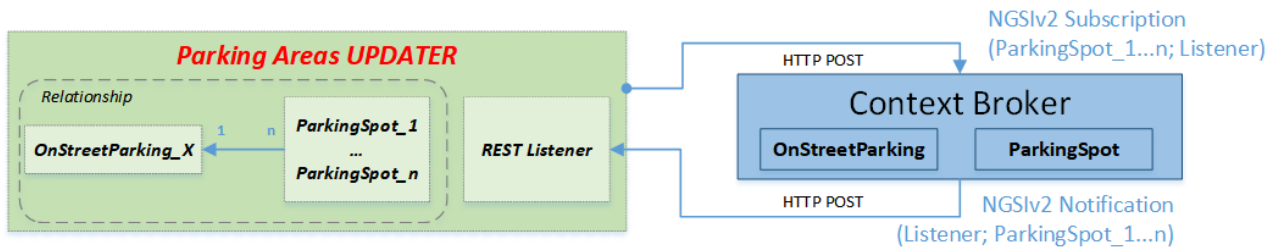
Communications between service components will be driven by the NGSI specification, using NGSI methods. The developments will rely on FIWARE NGSIv2 implementation of NGSI. This uses a RESTful API that provides access to all required NGSI methods:

²⁸ <http://fiware-datamodels.readthedocs.io/en/latest/Weather/WeatherForecast/doc/spec/index.html>

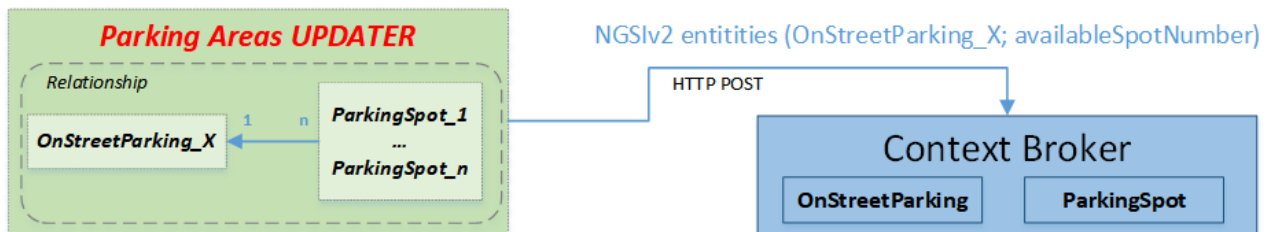
²⁹ <http://fiware-datamodels.readthedocs.io/en/latest/Weather/WeatherObserved/doc/spec/index.html>

³⁰ <http://fiware-datamodels.readthedocs.io/en/latest/Parking/OffStreetParking/doc/spec/index.html>

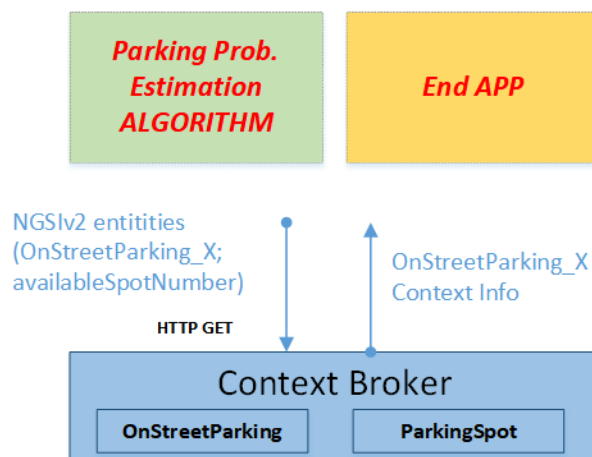
- **ParkingSpot subscription:** through this operation, the Parking Areas Updater is notified when a ParkingSpot entity changes its status. This operation subscribes (**NGSI Subscription**) each OnStreetParking entity to the set of ParkingSpots that forms it. Every time a notification (**NGSI notification**) is received, the Updater block updates the corresponding OnStreetParking entity.



- **OnStreetParking Status Update:** the Parking Areas Updater sends a **NGSI update** call to increase/decrease the number of available parking spots within the affected OnStreetParking entities.



- **OnStreetParking current status Request:** based on **NGSI queryContext** and NGSIv2 query entities implementation, the estimation algorithm building block and the final application (e.g. the multimodal navigator) can obtain the current status (num. of free parking spots) of selected OnStreetParking entity.



- **OnStreetParking historical data Request:** historical data requests will be done through the Synchronicity Historical API Data access [Ref D2.10]. These requests will provide the status (free/occupied parking lots) of the registered OnStreetParking entities (and/or OffStreetParking ones, if required) for selected time slots in the past.

2.3.4.3 Non-functional requirements.

It is also convenient to cover and consider a brief series of non-functional requirements that may have an impact over the offered service, even though their compliance or not should not compromise it. All in all, a couple of them are listed below:

- **Parking Sensors IoT deployment:** in order to deploy this service, the RZ's scenario **MUST** provide data related to parking spots monitoring. This is, the RZ should provide a Parking IoT deployments which includes, at least, information about:
 - Parking Spot location (geo JSON)
 - Parking Spot status (free/occupied)
 - Timestamp for each status reported (when has happen the status update)
- **City areas definition:** the RZ (by default) should group the monitored parking spots in city areas, mapped in OnStreetParking entities.

2.3.5 Implementation roadmap

The process to have this service ready and functional will comprise four different stages, as shown in Figure 10. The results of each of the phases will form the starting point of the next one; altogether, they will all build a structured approach to validating, identifying, prioritizing and planning the shaping and realization of this initial vision.

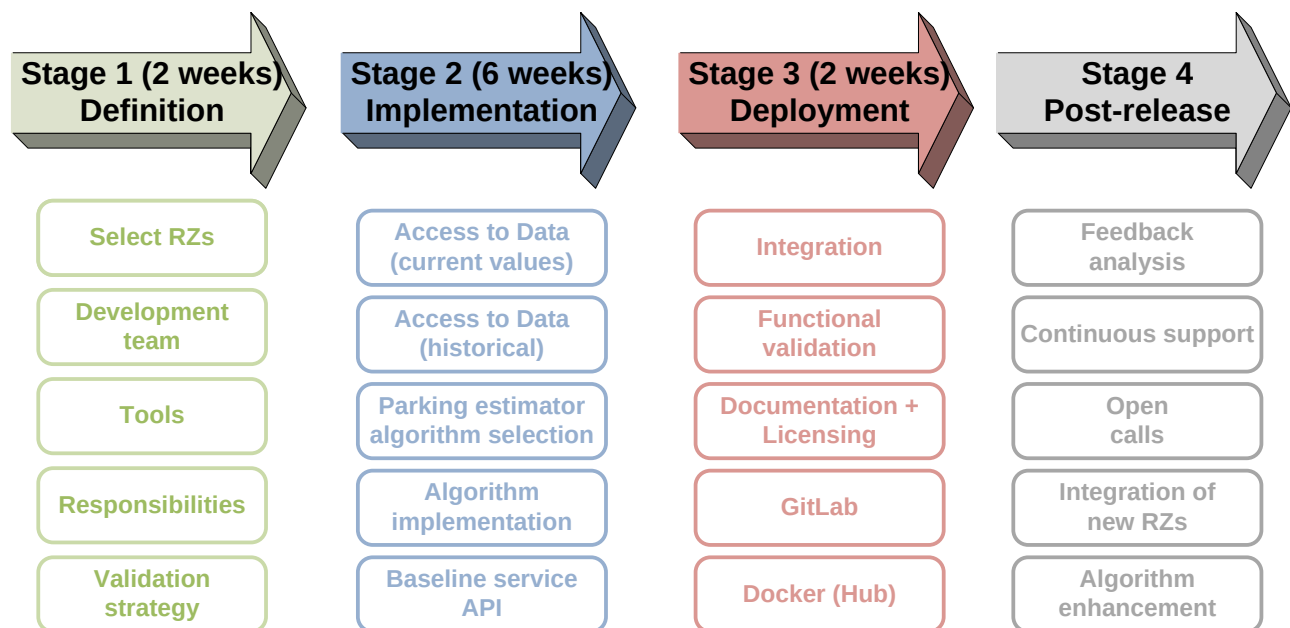


Figure 10 Parking estimator implementation roadmap (basic service)

During the first phase, we will define the set of RZs that will “integrate” their assets on this baseline service. It goes without saying that this means that they must fulfill all the requirements defined in WP2. At the same time, RZ representatives (and external developers from the consortium) will need to designate a team in charge of conducting the programming duties, which in turn will select the tools to carry out their work: programming language, libraries, IDE (Integrated Development

Environment), etc. Later on, a set of programming tools must be chosen and the programming team's hierarchies and responsibilities defined, to seamlessly integrate the results produced.

At the moment this stage is completed, the (development) team can proceed with the actual implementation process, where the API request methods and the way to retrieve and use data are properly implemented. The initial version of the required algorithm will be also sketched at this phase. In layman's terms, this "preliminary" phase will be followed by a basic analysis, where we will rely on the current status of and the historical patterns of the different parking sensors. With this information, the algorithm will yield a rather simple estimation of the probability of finding a free parking lot in a particular area. Together with this, it is worth highlighting that this phase also embraces a thorough state-of-the-art analysis of the current (and open source) solutions available in the literature that might help us implement an accurate estimation algorithm based on a demonstrated analytical model.

After the implementation period, it is time to focus on how we are exposing the service to the external world. Following the methodology described in (SynchroniCity, D3.4, 2018), this phase spans various steps: 1- integration process; 2- a functional validation check where all modules and operations must pass in order to ensure to proper execution on other systems; 3- the documentation composition (which has to accomplish some quality standards); 4- following an open-source philosophy, the source code will be published on Synchronicity's GitLab repository³¹; 5- the last step corresponds to the creation of a Docker image (which will be exposed in Synchronicity's Docker Hub Marketplace³²), so that any person could make a clean and plug-and-play installation process.

In parallel to these stages, the involved partners will be also responsible for giving both technical and non-technical support to issues that may arise from either internal or external people. Besides, this might also lead to the addition of new RZs as part of the baseline service's "catalogue". Likewise step 3, the methodology for this is explained in depth in (SynchroniCity, D3.4, 2018).

Last, but not least it is worth highlighting that this phase corresponds to the first version of the "Parking probability estimation" baseline service, and we do plan a plan beyond. This said, the next version of this deliverable will gather the next steps followed to enhance the set of features supported by this service, spanning the adoption/integration of more RZs, the improvement of the estimation algorithm by including heterogeneous data (events calendar, traffic analysis, weather forecasts, etc.).

³¹ <https://gitlab.com/synchronicity-iot>

³² <https://hub.docker.com/u/synchronicityiot/>

2.4 Traffic flow estimator baseline service

2.4.1 Description

When going from point A to B, one important item to consider is the time needed to get there. And, in urban context, traffic status is probably one of the most determinant aspects to address it.

Looking at the current traffic status may not be the best approach when trying to figure out how long it will take to get to point B.

A given street can be currently without traffic but, by the time the user gets there, the traffic flow may be very different.

To mitigate the problem, this service will analyze context data (based on TrafficFlowObserved³³ entities, currently available in SAN and POR) to provide current (close to real-time) traffic status and exploit the historical data to estimate traffic status.

This could be used just for information presented to the user or to feed the route calculation service in order to add a 'cost/weight' to each street when picked by the routing algorithm.

Out of the scope of the multimodal application, this service could help take actions like dispatch counter measures to alleviate traffic, advise users on the routes to avoid streets with heavy traffic, or even use it to raise alerts in case a predicted situation is very different from the actual one (let us say that if the service predicts heavy traffic on a street, but it is currently with very light traffic, then some disruption may have happened).

2.4.2 Application theme Functional Requirement addressed

FR code	FR description	Comments
FR-26	Traffic sensors must at least provide their GPS location and the traffic flow intensity.	Most of the traffic sensors used by RZs are providing only the traffic intensity of the exact location statically mapped by the respective RZs' administrator. The estimation of the traffic flow intensity from a point to another is, instead, addressed by the Traffic flow estimator baseline service.

Table 20 Multi-modal transportation application theme FRs addressed

2.4.3 Interface

2.4.3.1 Traffic Service API

This section describes a draft version of the Traffic Service API; it is important to underline that this is the first version of the API and that the implementation may differ from the description reported in this document. The final version of the APIs will be provided in the next iteration of this deliverable (SynchroniCity, D3.3, 2018).

³³ <http://fiware-datamodels.readthedocs.io/en/latest/Transportation/TrafficFlowObserved/doc/spec/>

The Traffic Service APIs will provide information about traffic status in streets and roads belonging to a city, in terms of the real time status of traffic and of the expected one in a given time slot, based on the current and historical conditions.

The first version of Traffic Service APIs consists of three RESTful services:

- **List Traffic Flows:** this API will return the list of the registered Traffic Flow observations; it is described in Table 21.
- **Current Traffic Flow Conditions:** this API will return the current conditions of the specific observed Traffic Flow; it is described in Table 22.
- **Traffic Flow Conditions Estimation:** this API will return the estimation of the traffic conditions of a specific observed Traffic Flow in a specific time of the day; it is described in Table 23.

It is important to underline that *List Traffic Flows* and *Current Traffic Flow Conditions* APIs provide a more easy and contextualized way to retrieve information from the context broker managing all the entities used by the service to provide its functionalities; indeed it will be possible to retrieve the same information by the use of the native APIs of the context broker, for instance the context broker adopted by a RZ.

Furthermore, a short description of data structures adopted in the APIs is also reported in this section.

The draft technical description of the Traffic Service APIs is also available at <https://synchronicitytrafficservice.docs.apiary.io/>

API Name		List Traffic Flows			
Description		This service will return the paginated list of the registered observed Traffic Flows.			
End-Point		/trafficflows{?rows,offset,type}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
rows	Query	number	No	20	This parameter is used to paginate the results and represents the first element of the result list where to start.
offset	Query	number	No	0	This parameter is used to paginate the results and represents the first element of the result list where to start.
Responses					
Code		Content-type	Entity		Description
200		application/json	Array of TrafficFlowObserved		The list of the observed Traffic Flows.
400		application/json	ErrorResponse		Bad Request (e.g. rows parameter not numerical)

500	application/json	ErrorResponse	Server error

Table 21 List Traffic Flows API

API Name		Current Traffic Flow Conditions			
Description		This service will return the current conditions of the specific observed Traffic Flow identified by the parameter <i>trafficflowID</i> . Each observed Traffic Flow is represented by an instance of the data model <i>TrafficFlowObserved</i> .			
End-Point		/trafficflows/{trafficflowID}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
trafficflowID	Path	String	Yes	-	The unique identifier of the observed traffic flow.
Responses					
Code		Content-type	Entity		Description
200		application/json	TrafficFlowObserved		The requested observed Traffic Flow
404		application/json	ErrorResponse		Observed Traffic Flow not found
500		application/json	ErrorResponse		Server error

Table 22 Current Traffic Flow Conditions API

API Name		Traffic Flow Conditions Estimation			
Description		This service will return the estimation of the traffic conditions of a specific observed Traffic Flow specified by the parameter <i>trafficflowID</i> in a certain time specified by the parameter <i>estimationTime</i> , taking advantage of the historical traffic conditions of the observed Traffic Flow. The query parameter <i>estimationTime</i> is mandatory.			
End-Point		/trafficflows/{trafficflowID}/estimatedconditions{?expectedArrivalTime}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
trafficflowID	Path	String	Yes	-	The unique identifier of the observed traffic flow.

estimationTime	Query	String	Yes	-	ISO8601 date format representation of the time for which the estimation is requested.
Responses					
Code	Content-type	Entity			Description
200	application/json	TrafficFlowConditionEstimation			The estimation of the traffic condition.
404	application/json	ErrorResponse			Observed Traffic Flow not found
500	application/json	ErrorResponse			Server error

Table 23 Traffic Flow Conditions Estimation API

2.4.3.2 Information Model

This section provides the description of the entities used in the Traffic Service APIs.

The entities reported in this section are:

- ErrorResponse: this entity provides error messages that may occur during the execution of a process; its description is reported in Table 24.
- TrafficFlowObserved: this entity models an observed traffic flow; its description is reported in Table 25.
- TrafficFlowConditionEstimation: this entity models the estimation of the traffic condition for an observed traffic flow in a certain time of the day; its description is reported in Table 26.

Name	ErrorResponse	
Description	This entity is used to provide information about errors occurred during the execution of a request through the Traffic Service APIs. Error codes and messages will be defined during the project. Specific error codes and relative messages will be defined later.	
Attributes		
Name	Type	Description
code	String	A string that represents the specific platform error code.
message	String	A string with the specific platform error message.

Table 24 ErrorResponse Entity

Name	TrafficFlowObserved	
Description	This entity is a generic object that will contains information about observed traffic flow entities and represents an observation of the condition of traffic flow in a specific place and time. Its main attributes are reported in this table; for more details, please refer to TrafficFlowObserved FIWARE data model ³⁴	
Attributes		
Name	Type	Description
id	String	A string that represent the unique identifier of the observed traffic flow.
location	Object	Location of the observed traffic flow represented by a GeoJSON.
dateObserved	DateTime	The date and time of the observation in ISO8601 UTC format.
intensity	Number	Number of vehicles detected in the observation.

Table 25 TrafficFlowObserved Entity

Name	TrafficFlowConditionEstimation	
Description	This entity models the traffic flow condition estimation in a certain time of the day.	
Attributes		
Name	Type	Description
congested	Enum (Red, Yellow, Green)	Indicates whether there will be a possible traffic congestion using three different indicators: • Red: very congested • Yellow: congested • Green: not congested
intensity	number	Estimated total number of vehicles
description	String	Describe how the estimation is made (e.g.: “Estimation based on Historical Data, weather forecast, events happening on the area”)

Table 26 TrafficFlowConditionEstimation Entity

2.4.3.3 Interaction diagram (with the external world)

Figure 11 shows the possible usage of the baseline service.

³⁴ <http://fiware-datamodels.readthedocs.io/en/latest/Transportation/TrafficFlowObserved/doc/spec/index.html>

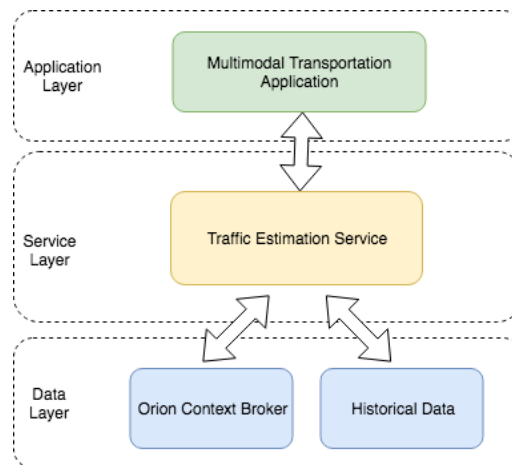
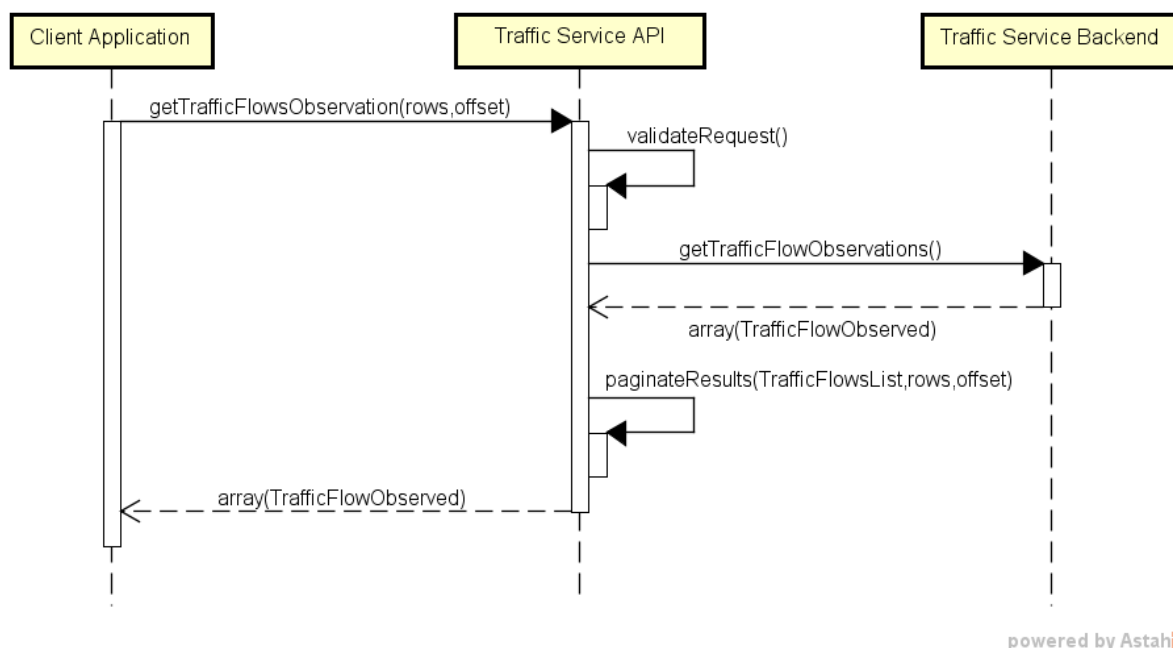


Figure 11 Traffic flow estimator baseline service interaction diagram

The traffic estimator can either be accessed by the application itself, or by the route calculator service (in order to get a cost for each segment).

Interaction through 'List Traffic Flow'

This service provides a list of the observed Traffic Flows registered in the platform; the provided list is paginated: the *rows* and *offset* parameters are used to manage the pagination. The expected interaction is depicted in Figure 12.



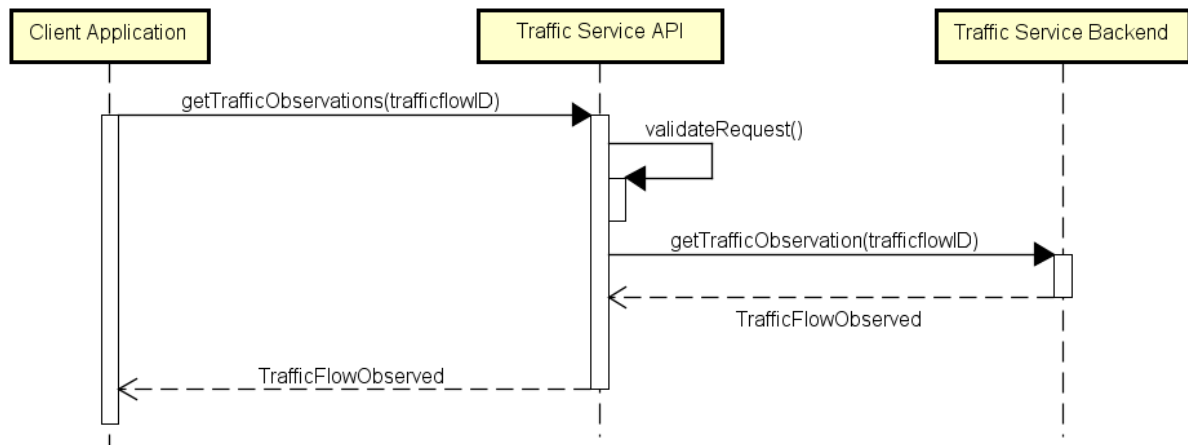
powered by Astah

Figure 12 List Traffic Flow

The Client Application requests for the registered observed traffic flows to the Traffic Service API, this component will validate the request and forward it to the Traffic Service Backend who will retrieve the registered observed traffic flows. The resulting list will be paginated and returned to the Client Application.

Interaction through 'Current Traffic Flow Conditions API'

This API will return the detail of a specific observed traffic flow identified by the *trafficflowID* parameter. Figure 13 illustrates the interactions among components.



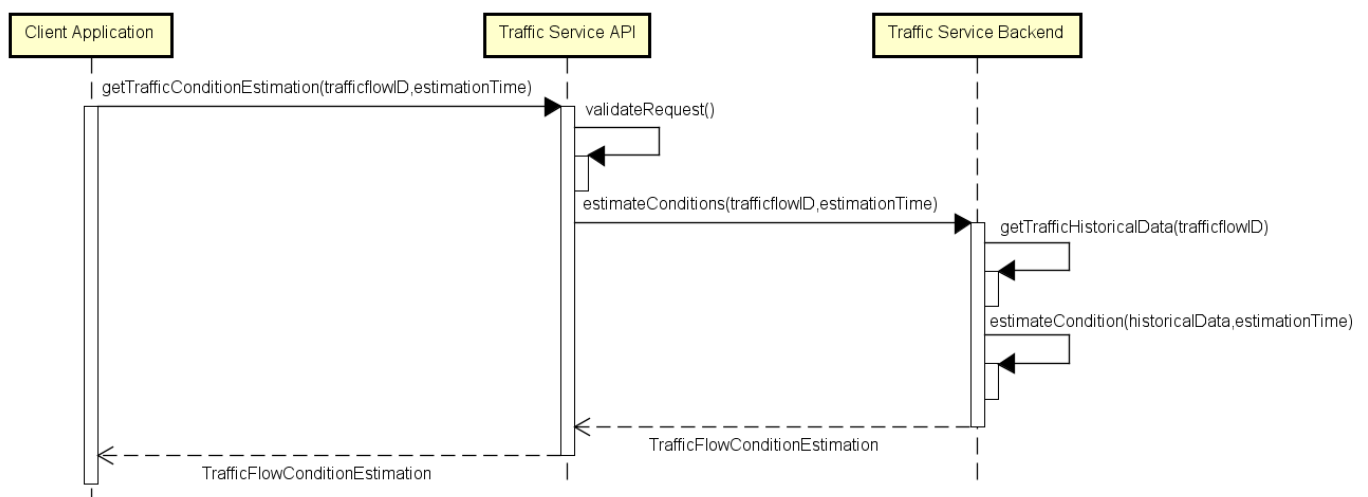
powered by Astah

Figure 13 : Current Traffic Flow Conditions

The Client Application request for the details of an observed traffic flow registered in the platform. The Traffic Service API will validate the request and forward it to the Traffic Service Backend. Then the Traffic Service Backend will return the requested observed traffic flow to the Traffic Service API. This component will forward the response to the Client Application.

Traffic Flow Conditions Estimation

This service will use observed both traffic flow current and historical data in order to provide an estimation of the conditions of the traffic flows in a specified time of the day, for a specific observed traffic flow. The flow of operations is showed in Figure 14.



powered by Astah

Figure 14 Traffic Flow Conditions Estimation

The Client Application requests for condition estimation of a specific observed traffic flow identified by the parameter *trafficflowID*, whereas the parameter *estimationTime* represents the expected day

and time for which the estimation is requested. The Traffic Service API will validate the request and will forward it to the Traffic Service Backend. This component will retrieve the historical data of the requested observed traffic flow and will use this information and the *estimationTime* to estimate the condition of the specific traffic flow. The result will be provided to the Client Application through the Traffic Service API.

2.4.4 Service Architecture

2.4.4.1 Service components

Figure 15 shows the architecture for the service, with all its building blocks, described below:

- REST API
 - o Handle interaction with client service
- Input Processing
 - o Make sure it's a valid request, enrich the request with context data, etc.
- Cache
 - o Responsible for caching results. This will improve response times whilst reducing load to the estimator algorithm, access to Context Broker and Historical Data Component. For flexibility, this component can be overridden or purged.
- Output Processing
 - o Based on input parameters, it may return a number (quantified result) or a qualified result ('cost/weight', red/yellow/green status, etc.).
- Estimation Algorithm
 - o Gets current data (from the context broker) and historical data from FIWARE components and computes an estimate, based on the input parameters.

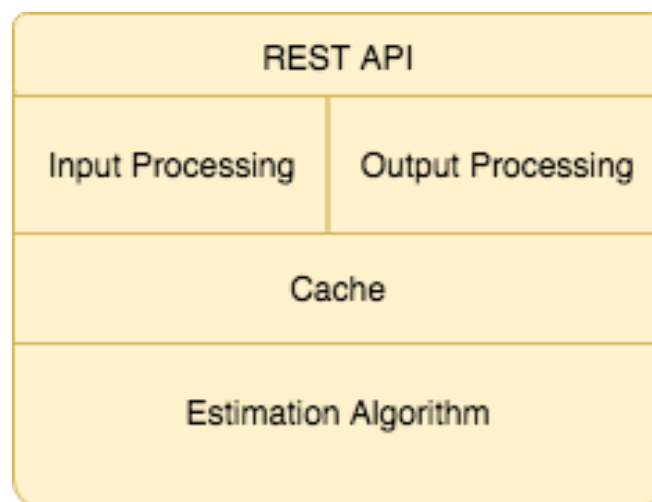


Figure 15 Traffic flow estimator service components

As for the entities, the service will require, from:

- SynchroniCity Context Manager component: through NGSI interface to provide current status (and location) of the following:

-
- o *TrafficFlowObserved*³⁵ (mandatory)
 - o *Alert*³⁶ (optional)
 - o *PointOfInterest*³⁷ (optional)
 - o *Events*³⁸ (optional)
 - o *WeatherObserved*³⁹ (optional)
 - o *WeatherForecast*⁴⁰ (optional)
 - SynchroniCity Historical Data component: through Historical Data API to provide the following entities historical data to analyse:
 - o *TrafficFlowObserved* (mandatory)
 - o *Alert* (optional)
 - o *PointOfInterest* (optional)
 - o *Events* (optional)
 - o *WeatherObserved* (optional)

2.4.4.2 Operation model (Message flows between components)

As shown in Figure 16 a typical message flow passing through the Traffic Flow estimation service foresees a HTTP POST Request (with JSON config payload) as input, which go through the internal processing, making leverage of caches when possible otherwise applying the estimation algorithm requesting the needed data through the NGSI and/or Historical API. The outcome is then returned within the synchronous response as a JSON message.

³⁵ <http://fiware-datamodels.readthedocs.io/en/latest/Transportation/TrafficFlowObserved/doc/spec/index.html>

³⁶ <http://fiware-datamodels.readthedocs.io/en/latest/Alert/doc/spec/index.html>

³⁷ <http://fiware-datamodels.readthedocs.io/en/latest/PointOfInterest/PointOfInterest/doc/spec/index.html>

³⁸ <http://schema.org/Event>

³⁹ <http://fiware-datamodels.readthedocs.io/en/latest/Weather/WeatherObserved/doc/spec/index.html>

⁴⁰ <http://fiware-datamodels.readthedocs.io/en/latest/Weather/WeatherForecast/doc/spec/index.html>

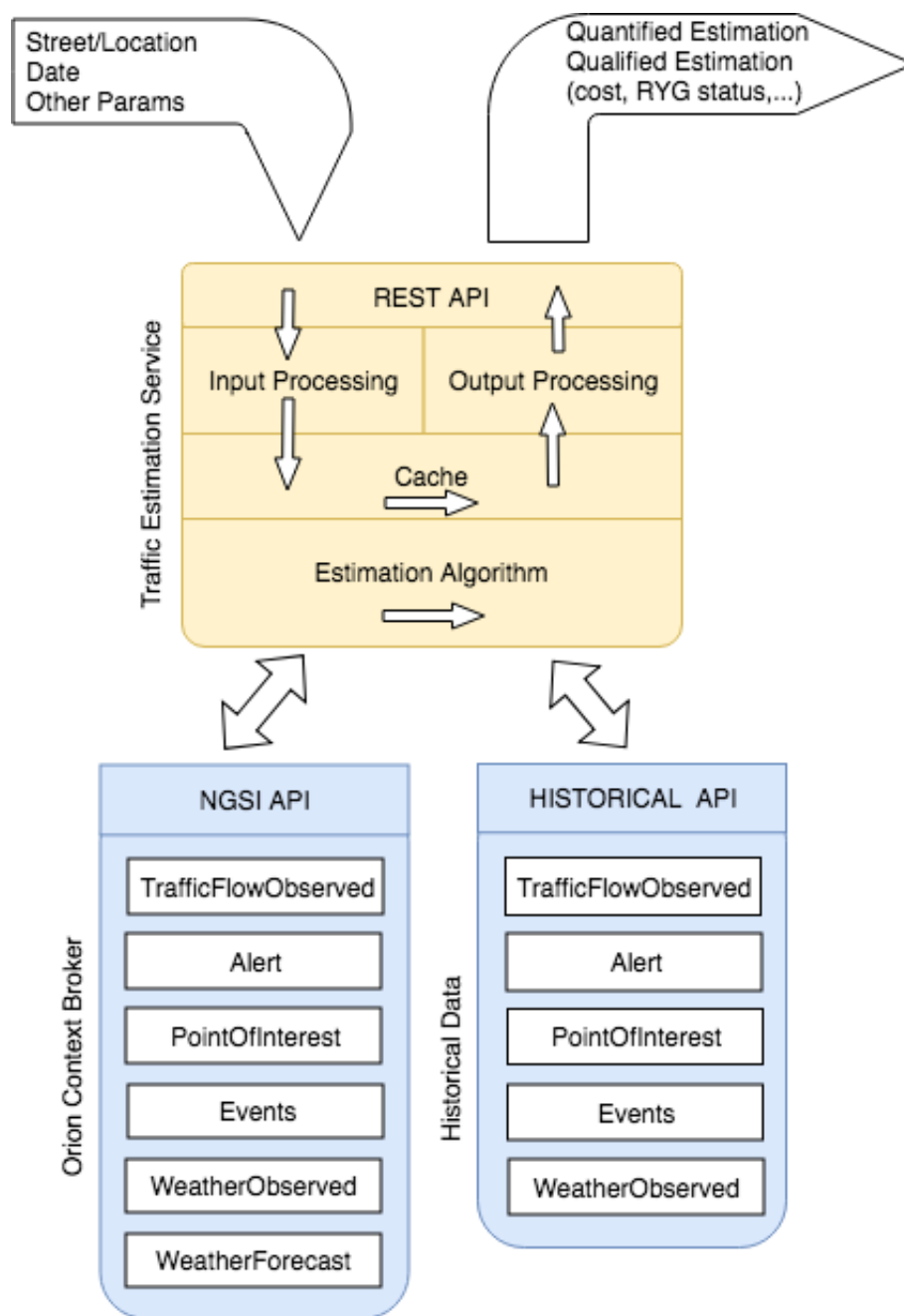


Figure 16 Traffic Flow Estimator service message flow

2.4.5 Implementation roadmap

For implementing this service, a roadmap has been planned (see Figure 17):

- Phase 0 – Definition (Week 1)

- 1) Define programming team
- 2) Define the programming language and basic libs to use
- 3) Define programming tools and collaboration strategy
- 4) Define tests strategy
- 5) Definition of request methods, the input parameters and the output format.
- 6) Define estimation algorithm strategy

- Phase 1 - Basic Architecture Implementation (Week 2 to Week 3)

- 1) implement the API request methods
- 2) implement parameter checking (json format, ranges, ...)
- 3) implement data gathering internal structures (efficiently load data required for decision making, based on the request and input parameters)

- Phase 2 - Feature implementation (Week 4 to Week 10)

- 1) Implement the Estimation algorithm
 - a. Estimate traffic based on historical data (traffic levels and meteorological conditions)
 - i. Magnetic sensors: Number of vehicles | Date | Time | GPS coordinates
 - ii. Meteorological historical data @ city level (for example, rain, wind speed, temperature)
 - b. Estimate traffic based on weather forecast and real-time weather conditions
 - i. Weather forecast @ city level (for example, rain, wind speed, temperature)
 - ii. Meteorological sensors @ city level or more granular (for example, rain, wind speed, temperature)
 - c. Estimate traffic based on impacting events
 - i. Events (for example, football matches, music festivals, city festivities)
 - ii. Traffic constraining events (for example, roads closed)
- 2) Implement a cache

- Phase 3 (Integration) tests (from Week 11)

- 1) Tests (TBD)
- 2) Support

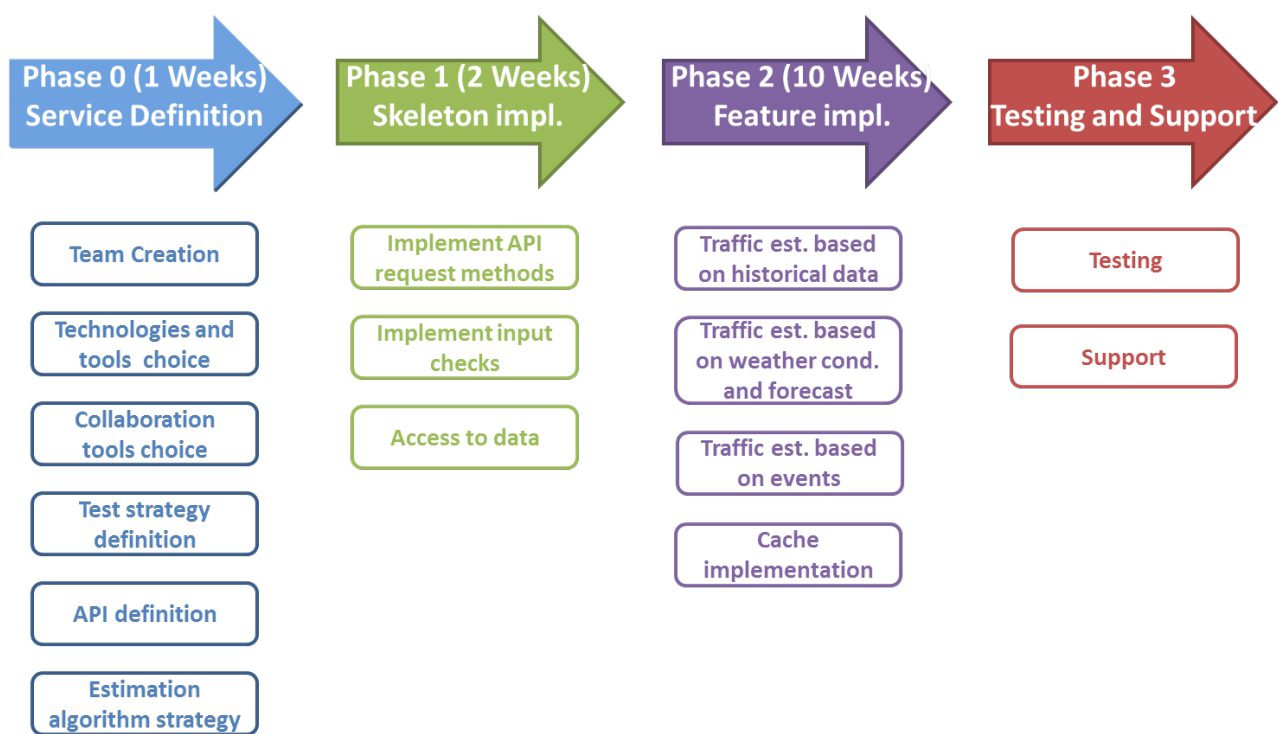


Figure 17 Gantt diagram of the Traffic Flow Estimation roadmap envisaging 10 weeks to achieve a first full implementation, without considering testing

2.5 Smartcities Dashboard service

2.5.1 Description

The Smartcities Dashboard tool is a web service interface that provides end-users a simple and user-friendly visualization of the devices that compose the SynchroniCity project in the different Reference Zones (RZs). Currently, the tool exposed is able to represent the location, real-time sensors information and relevant information presented as charts.

The current service version is relying on Google Maps API⁴¹ to create a web service that draws a map of the devices deployed along the world.

- All the devices are shown in real time, taking advantage of the Orion's publish/subscribe mechanisms. In this way, updated values for sensors and location are assured.
- The dashboard splits the data information between real time data and historical data. The first one is provided as soon as it is gathered. The second one is gathered and aggregated by the STH Comet.
- The dashboard has been created providing a well-defined user experience. Each point representing a device can be clicked deploying custom tabs regarding the sensors deployed.

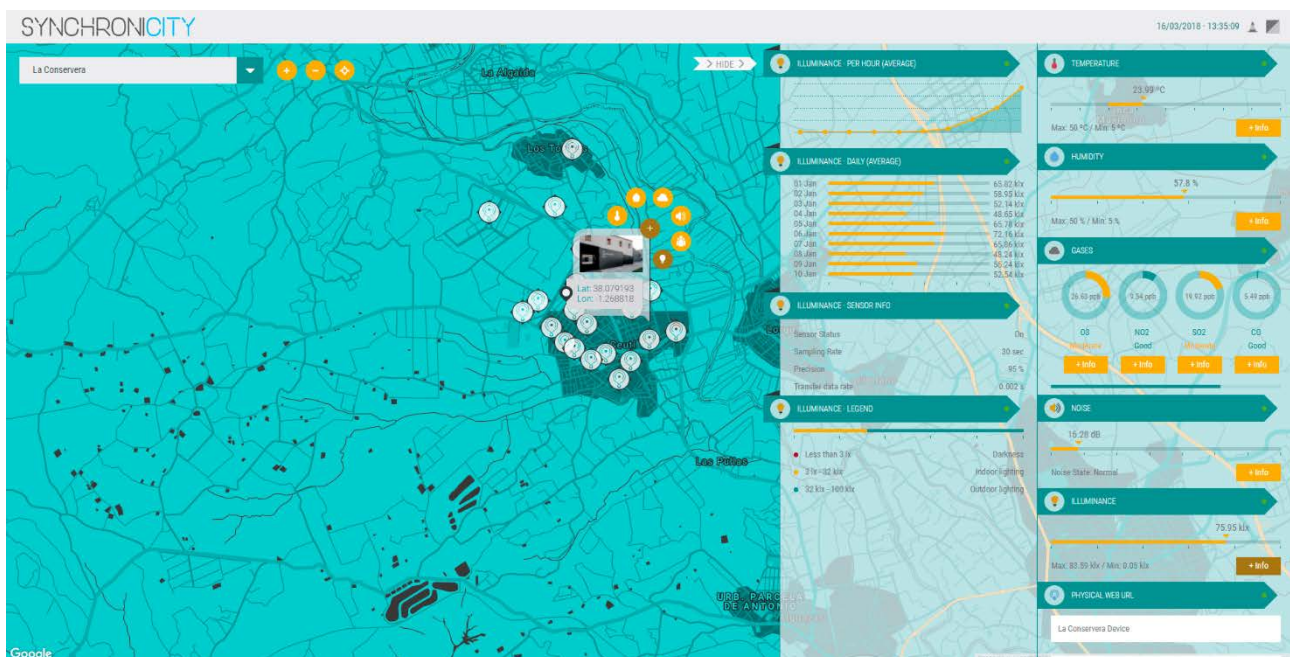


Figure 18 SmartCities Dashboard view

2.5.2 Application theme Functional Requirement addressed

FR code	FR description	Comments
---------	----------------	----------

⁴¹ <https://developers.google.com/maps/documentation/>

FR-1	System must process data from multiple sources/sensors in real-time	This visualization dashboard is capable of showing all the data available given as input.
FR-6	The system must provide flexibility in terms of viewing, or having access to, certain data sets (e.g. through dashboard widgets)	The dashboard is versatile and can be customized
FR-7	The system may provide a map as the main way of interaction to superimpose data sets	This dashboard service is capable of superimpose data.
FR-8	The system should provide different views on the superimposed data sets	This dashboard service offers many different views.

Table 27 FRs of the Human-Centric Traffic Management application theme addressed by the Smartcities Dashboard baseline service

FR code	FR description	Comments
FR-12	The user should be able to view on a map the points of interest located in an area near to the route.	POIs can be visualized in the map
FR-16	The application must provide users with a georeferenced search of parking areas.	Changing the focus on the map affects the geographic scope of the query request.
FR-18	The application must be able to indicate which transport possibilities are currently available near the selected parking area (e.g. estimated arrival times for nearby bus lines).	The data available, such as public transportation stops and estimated arrival times, can be shown on the map
FR-19	The application should be able to indicate in real-time where free parking lots Users, Parking are available and at which cost/minute.	The data available, such as parking spots status and parking costs, can be shown on the map

Table 28 FRs of the Multi-Modal Transportation application theme addressed by the Smartcities Dashboard baseline service

FR code	FR description	Comments
FR-4	UI requirements; system should be intuitive and easy to use	Accepted as input by the service

Table 29 FRs of the Community Policy Suite application theme addressed by the Smartcities Dashboard baseline service

2.5.3 Interface

2.5.3.1 Dashboard functionalities

In order to bring a better understanding of the Smartcities Dashboard capabilities, a description is provided below. In

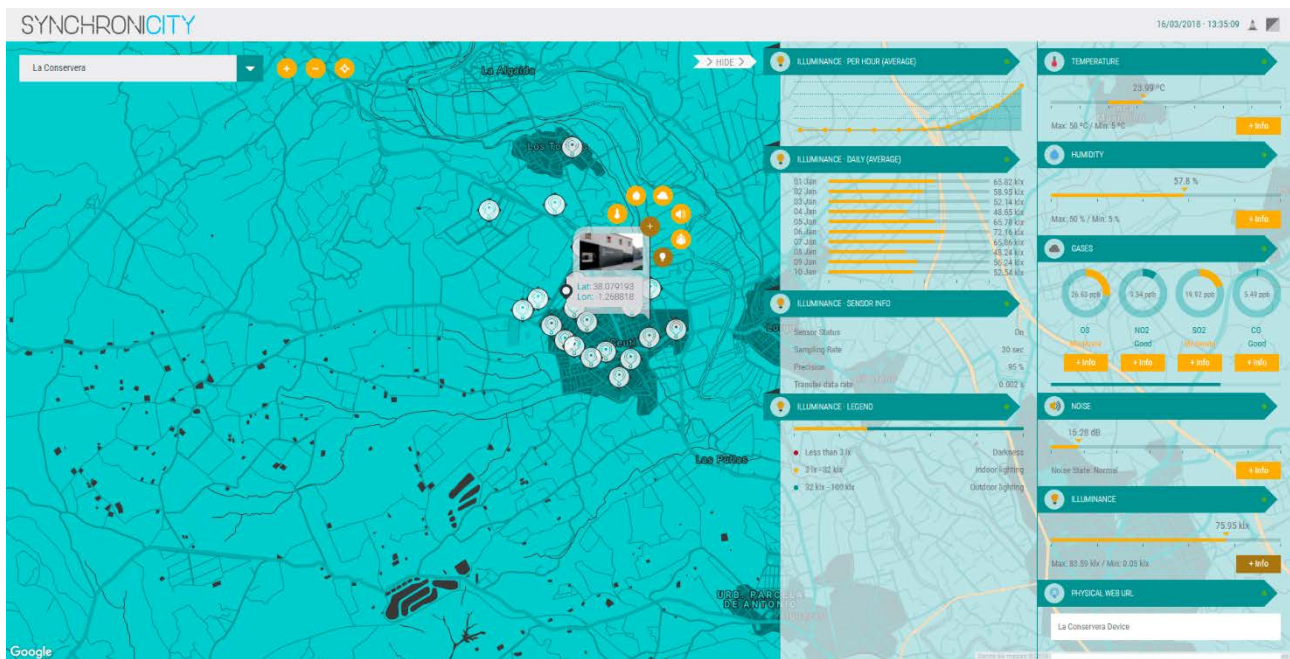


Figure 18 it is possible to see the functionalities below:

- Graphical display of devices or services, registered in Orion, based on a custom Google Maps visualization. The map visualization of devices is the main functionality covered by the solution. The solution brings a customizable map in which devices icons and map color can be changed.
- Device selector in a scrollbar form. A device selector list is provided in order to simplify the lookup of the devices. A mobile device can be selected even when its position is unknown using a human readable name.
- A device icon OnClick response providing location, a representative image, and links to sensors implementation. When a device is selected a custom information window is displayed providing relevant device's information.
- When a device is selected a right panel is shown with real-time information of the device. The solution brings real-time information for each sensor module implemented. Current modules are temperature, humidity, gases, noise, crowd monitoring, illuminance or physical web. The front-end architecture allows the inclusion of new modules.
- For each sensor module implemented in the device, an advanced panel is provided by clicking the +Info button. In this panel information like the last hour and last days average values are provided, besides this, a legend is provided to facilitate the value reads.

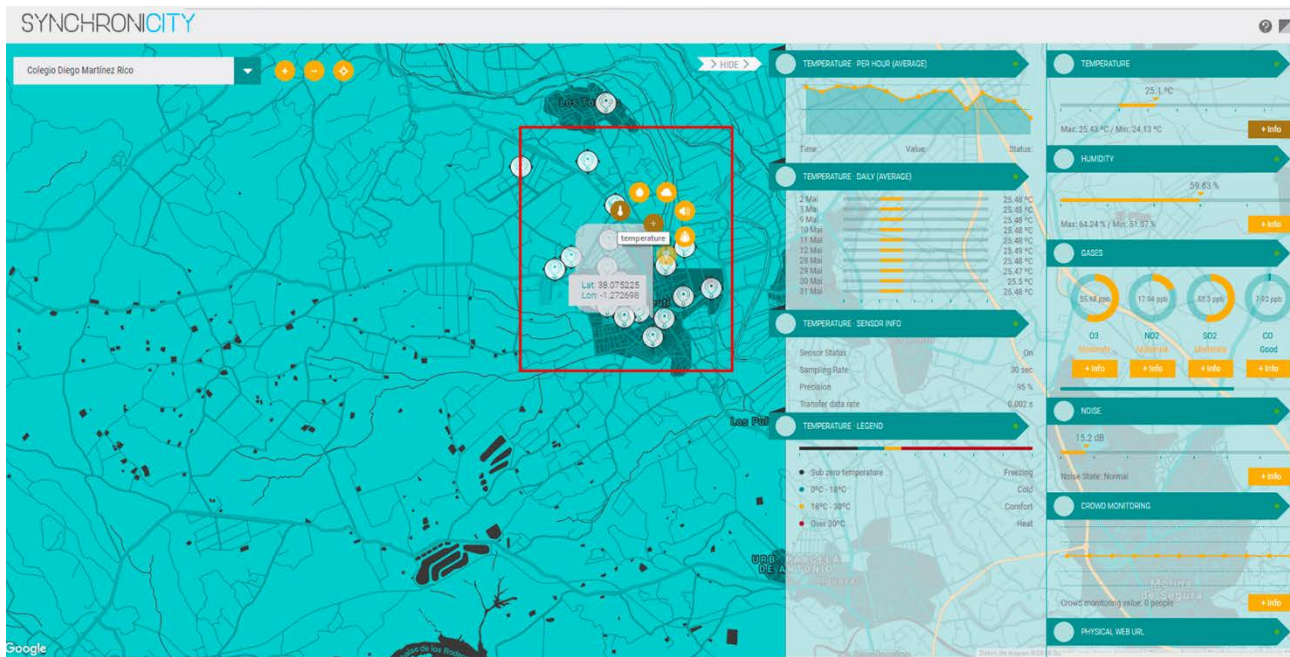


Figure 19 Advanced panel for visualizing device information

2.5.3.2 Backend server API

The backed server API can be accessed for getting the last value or the historical data.

- **Last value:** GET /api/v1/real-time-data → retrieve the devices stored as entities in Orion Context Broker (Response 200 OK).

```
[
  {
    "id": "HOP240ac406551e:SmartSpot",
    "type": "SmartSpot",
    "area": 10,
    "availability": "T9-10",
    "bluetoothChannel": "37",
    "interval": 500,
    "physicalUrl": "https://goo.gl/jMMgf8",
    "signalStrength": "highest"
  },
  ...
]
```

- **Historical data:** GET /api/v1/historical-data/<device_id> → retrieve advanced information from STH Comet related to device with device_id (Response 200 OK)

```

{
  "<attribute>": [ // attributes may be any sensor connected: temperature, humidity, etc.
    {
      operation: [ min | max | average_day | average_month ],
      value: Float | [ {"value": Float, "timestamp": String }
    ], // Timestamp could be hour if average_day or day/month otherwise
    ...
  ],
  ...
}

```

2.5.3.3 Information Model

The information model in the southbound interface, that is the related FIWARE IoT Service, is OMA NGSI. Each device is represented as an Entity. The attributes contained in these entities can be configured by the Context Provider that registers this information.

In the northbound interface the information model can be seen in both responses in previous section.

2.5.3.4 Interaction diagram (with the external world)

In the diagram shown in Figure 20, two clearly differentiated layers can be observed. The FIWARE IoT stack layer that will be provided by the cities within the synchronicity ecosystem and the layer of the visualization tool of SynchroniCity that is deployed on the FIWARE layer and contains the necessary services to deploy the dashboard.

Below is a brief description of the services and how they interact with the system:

- **Smart Cities Dashboard:** This is a UI where users can see the devices available throughout a city and the values collected by them. In addition, the visualization tool has a section where the user can see the time series. This service is communicated with the Smart Cities Backend to acquire this data.
- **Smart Cities Backend:** It is responsible for collecting the data of the FIWARE services and sending them to the Dashboard to represent them. This service communicates directly with the Orion Context Broker and with the STH Comet that will make use of the services they depend on to satisfy the requests

2.5.4 Service Architecture

Currently, the service architecture, represented in Figure 20, is made up of a server that interacts directly with the STH Comet and Orion Context Broker FIWARE components, requesting devices' data, and a web page on which the data is represented, based on a custom map.

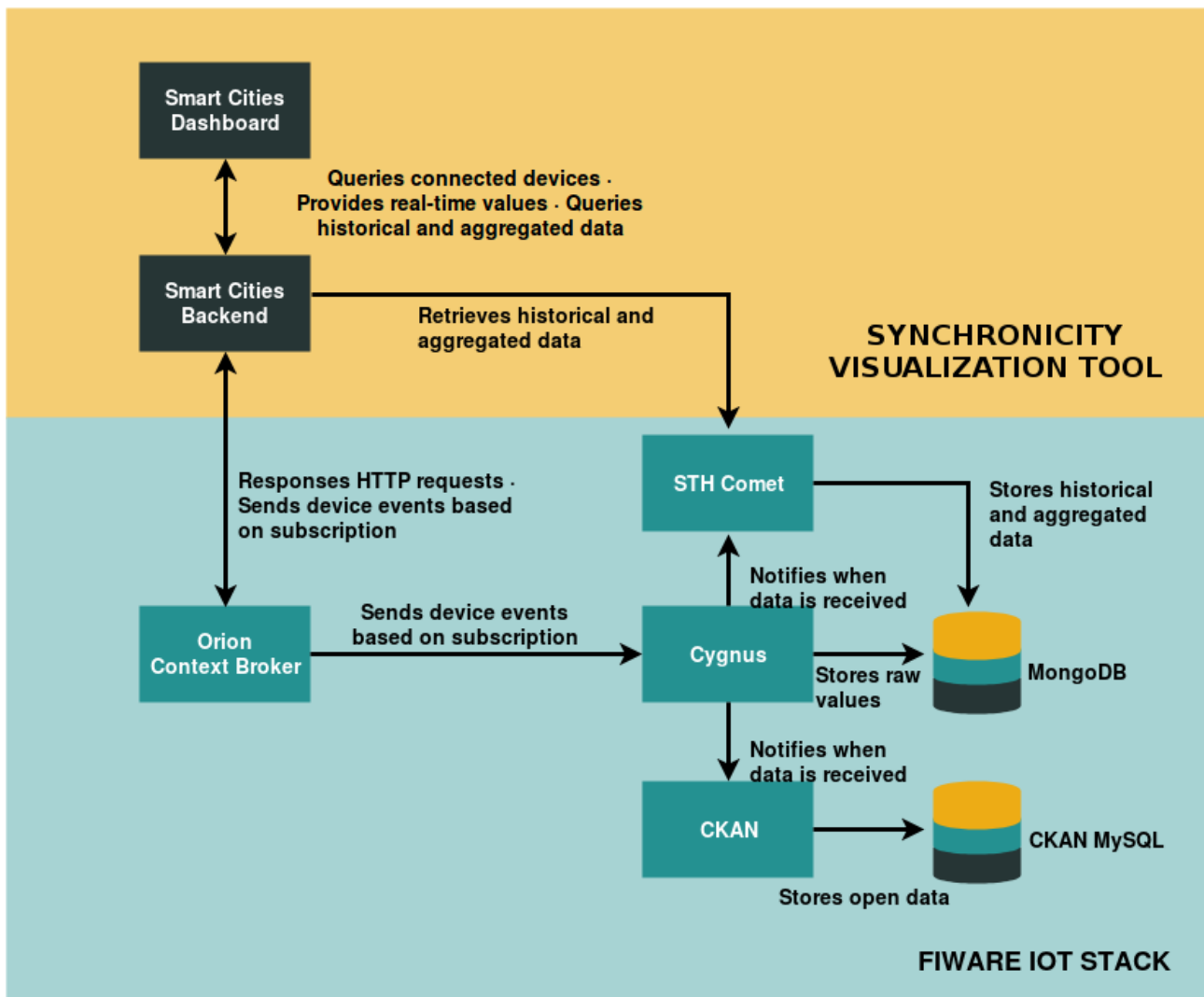


Figure 20 SmartCities Dashboard architecture

2.5.4.1 Service components

- FIWARE IoT Stack: the solution is based on the main FIWARE components.
 - o The device information is gathered through different IoT Agents regarding the IoT protocol used.
 - o This information is provided to Orion Context Broker. Orion will provide the last value information when is requested by the backend and, using its publish/subscribe mechanisms; Orion will notify Cygnus each time a value is received.
 - o Cygnus will gather and store this information according to the databases connected. In our case MongoDB and the STH Comet service.
 - o When requested, STH will provide historical and aggregated data of the different devices registered in Orion.
- SmartCities Backend: this service acts as a bridge requesting and adapting the information provided by Orion Context Broker and STH Comet and sending it to the frontend.
- SmartCities Frontend: this service provides a graphical interface to access and visualize the different devices. Map and chart representations are provided.

2.5.4.2 Operation model (Message flows between components)

Regardless the message flow between the FIWARE components, the SmartCities Dashboard solution components flows go from the frontend to the backend server and from this to Orion and STH Comet. All the traffic is synchronous and over HTTP protocol.

2.5.5 Service implementation

2.5.5.1 Service implementation IT environment (dependencies like SQL db)

The Smartcities Dashboard does not require any other components than the docker framework. In fact, once the two docker images of the “backend” server and the “frontend” dashboard are downloaded and instantiated, the baseline service as an whole will interact with the SynchroniCity infrastructure.

2.5.5.2 Service deployment dependencies (e.g. Orion Broker, oAuth server etc.)

The Smartcities Dashboard baseline service requires basically four services from the SynchroniCity reference infrastructure:

- *FIWARE Orion Context Broker* which hosts the last observations of each resource and notifies data to interested subscribers.
- *FIWARE Cygnus*: that acts as connector to the MongoDB
- *FIWARE STH Comet* that handles historical queries.
- *MongoDB* which is the persistent database that is in charge of storing data.

2.5.5.3 Installation and admin guide

The services are dockerized, different deployment files are provided and also a deployment guide is being written⁴².

⁴² <https://github.com/HOP-Ubiquitous/live-dashboard/blob/master/README.md>

2.6 Metrics Visualizer (Grafana) service

2.6.1 Description

Metrics Visualizer is a custom deployment of Grafana⁴³ software working on top of FIWARE Stack services.

Through Grafana, a set of databases can be configured as datasources in order to create personalized charts and tables. These databases are filled using QuantumLeap⁴⁴ or FIWARE Cygnus as device information subscriber..

Grafana provides user management system allowing the creation and manage of different isolated dashboards with custom charts and device information. An example of a dashboard created with Grafana can be seen in Figure 21. .

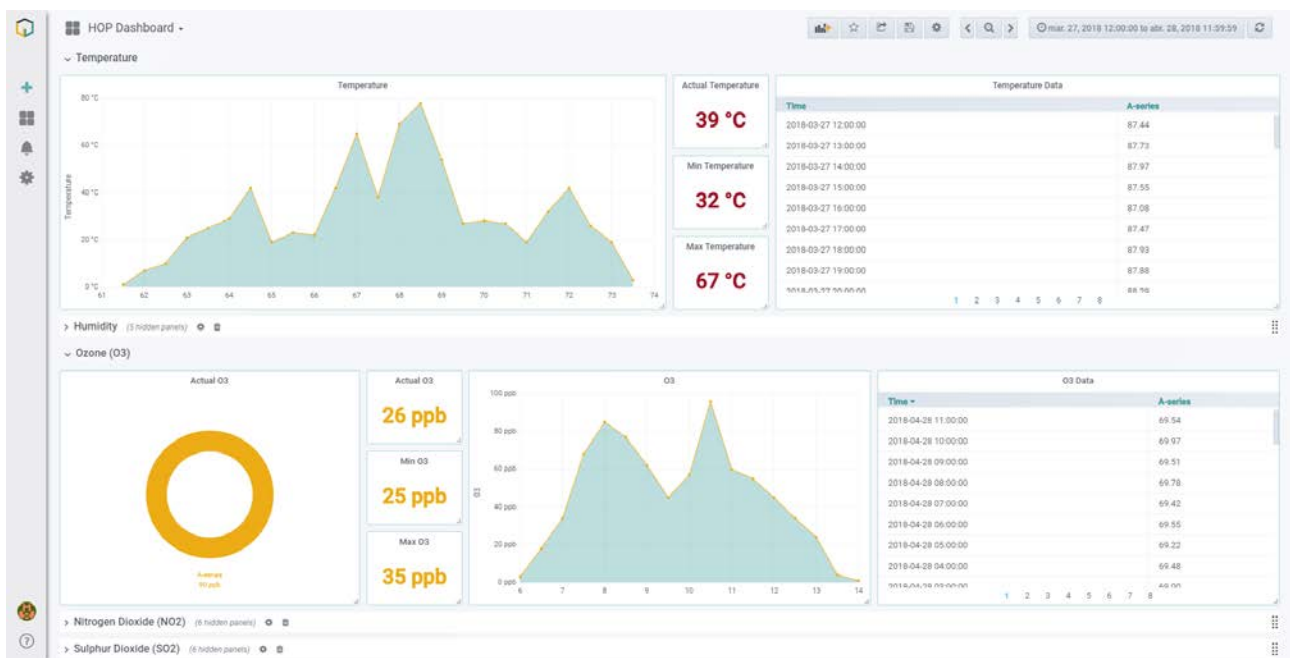


Figure 21 Grafana Dashboard example

2.6.2 Application theme Functional Requirement addressed

FR code	FR description	Comments
---------	----------------	----------

⁴³ <https://grafana.com/>

⁴⁴ <https://smartsdk.github.io/ngsi-timeseries-api/>

FR-1	System must process data from multiple sources/sensors in real-time	This metrics visualization is capable to show all the data available given as input and refresh the visualization on real time every time new entries are produced.
FR-6	The system must provide flexibility in terms of viewing, or having access to, certain data sets (e.g. through dashboard widgets)	The dashboard allows exporting the visualization using iframes in order to allow users to exploit the visualization using external or third party applications.
FR-10	The system should allow selection of specified time window and display corresponding aggregated data over the selected time window	The dashboard allows users selecting from a wide range of temporary windows and even creating custom time windows.
FR-11	The system may allow for annotation specified time windows	The dashboard allows user to customize desired annotations

Table 30 FRs of the Human-Centric Traffic Management application theme addressed by the Metric Visualizer (Grafana) baseline service

FR code	FR description	Comments
FR-2	Editable dashboards; based on city / team requirements user should be able to amend and lock (based on hierarchy) the dashboard "widgets"	The dashboard provides an authentication system based on organizations where users have different roles which will give them different levels of privileges limiting the ability to modify the dashboard
FR-3	Data management; system should be able to take custody of the data and edit for visualisation	The system supports to export the time series data using json or csv format in order to facilitate users to manipulate or apply advanced techniques of data analysis.
FR-4	UI requirements; system should be intuitive and easy to use	The system provides three levels of permissions (admin, edit and view) where users who have an edit rank or higher can generate boards in a simple way and adapt them to the new needs in an agile way.

Table 31 FRs of the Community Policy Suite application theme addressed by the Metric Visualizer (Grafana) baseline service

2.6.3 API

2.6.3.1 Interface

Information about the API provided by Grafana can be found in the official documentation of this tool, i.e., following the next url http://docs.grafana.org/http_api/dashboard/.

API access can be managed, allowing or denying access using the GUI dashboard, and, in the same way, security methods can be put in place.

2.6.3.2 Backend server API

As explained in the previous section, for more information about API and data models used, the official Grafana documentation might be consulted.

2.6.3.3 Information Model

The information model in the southbound interface, that is the related FIWARE IoT Service, is OMA NGSI. Each device is represented as an Entity. The attributes contained in these entities can be configured by the Context Provider that registers this information.

In the northbound interface the information model can be seen in both response in previous section.

2.6.3.4 Interaction diagram (with the external world)

Grafana is a tool that allows to visualize time series in an easy and intuitive way. For this purpose, Grafana makes use of data provided by databases prepared to store time series such as CrateDB⁴⁵. Since no FIWARE component was able to feed this type of database, the QuantumLeap component was created in the scope of the smart SDK project. This component is a service that transforms data in NGSI format to time series. QuantumLeap is published as an open source project⁴⁶.

Figure 24 shows how the necessary services interact between them in enable the functionalities of the Grafana visualization tool.

Grafana is a tool that allows us to visualize time series in an easy and intuitive way. For this purpose, Grafana makes use of data provided by databases prepared to store time series such as CrateDB. Since no FIWARE component was able to feed this type of database, the QuantumLeap component was created in the scope of the smart SDK project. This component is a service that allows transforming data in NGSI format to time series. QuantumLeap is published as an open source project⁴⁷.

2.6.4 Service Architecture

Currently the service architecture, shown in Figure 26, is made up of a custom Grafana deployment that interacts with CrateDB requesting devices data, and a web page in which the data is represented based on a custom dashboard.

Figure 22 Interaction Between Services

⁴⁵ <https://crate.io>

⁴⁶ <https://github.com/smartsdk/ngsi-timeseries-api>

⁴⁷ <https://github.com/smartsdk/ngsi-timeseries-api>

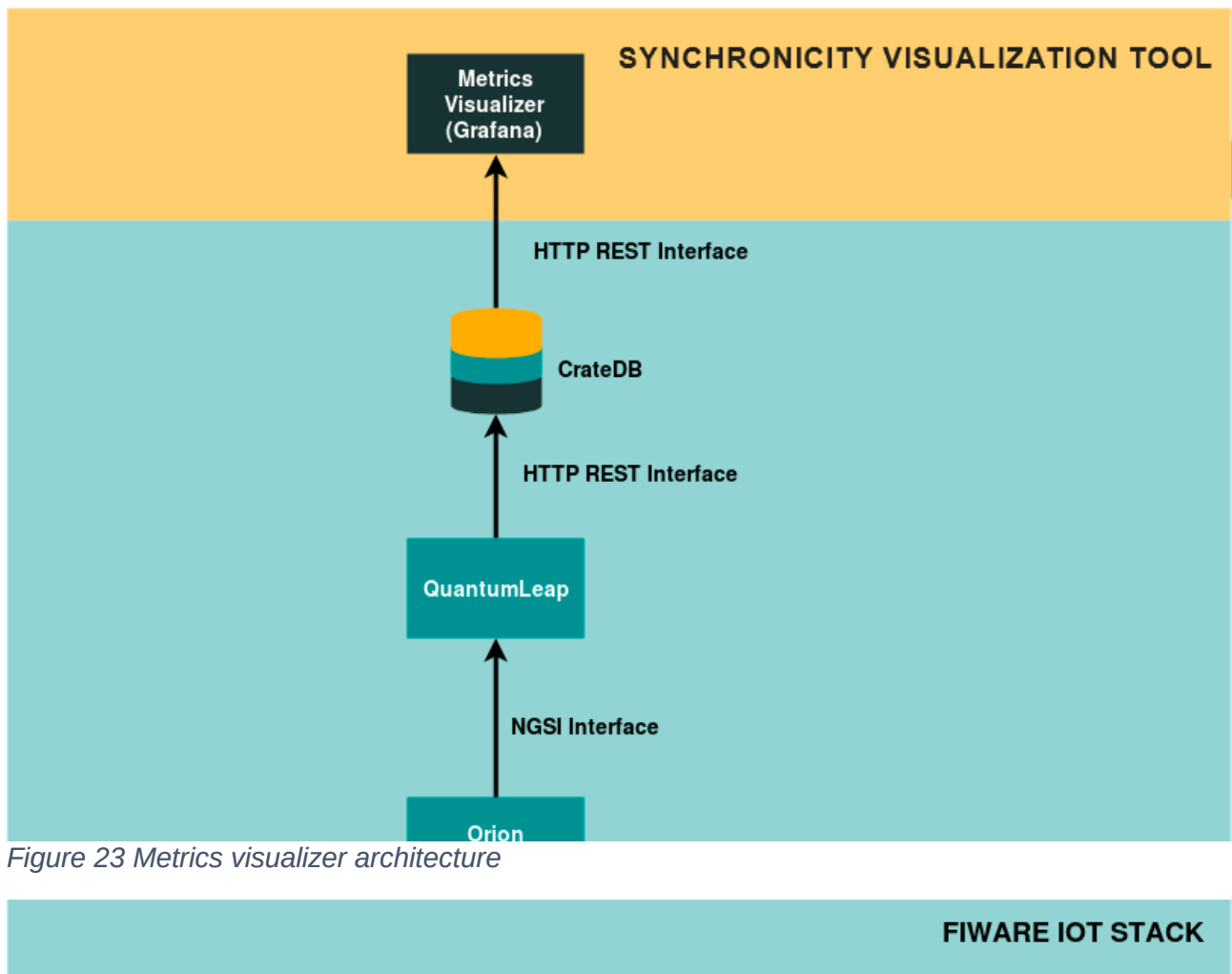


Figure 23 Metrics visualizer architecture

2.6.4.1 Service components

The components involved in the Metrics Visualizer service are grouped in two layers:

- **FIWARE IoT Stack:** the solution is based on the main FIWARE components. The data is provided by Orion Context Broker with the last value when requested and, using its publish/subscribe mechanism, Orion will notify QuantumLeap each time a value is received.
 - QuantumLeap will gather and store this information according to the databases connected. In our case CrateDB.
 - CrateDB will hold historical data.
- **Metrics Visualizer (Custom Grafana deployment):** data visualization and monitoring tool. The software provides an easy way to create custom dashboards. Dashboards are composed of individual Panels arranged on a grid. The software ships with a variety of Panels. The software makes it easy to construct the right queries, and customize the display properties so that you can create the perfect Dashboard for your need. Each Panel can interact with data from any configured data source.

Figure 24 Metrics visualizer architecture

2.6.4.2 Operation model (Message flows between components)

Information flow.

Grafana bases part of its power on the use of special databases in time series, since Orion context broker does not provide any type of adapter for this type of database, an external service has been utilized: QuantumLeap⁴⁸. It allows transforming the context elements stored in the Orion Context Broker into a format compatible with CrateDB. To be able to store the data generated in real time, it is necessary to make a subscription of the data wanted to monitor so that Orion Context Broker sends it to QuantumLeap which will store in CrateDB. Grafana is then able to understand the data structures of CrateDB, therefore it will be able to read directly from it and provide the user with tools to represent the data.

User interaction flow.

Grafana provides a simple and friendly user interface that allows users to acquire different roles and belong to different organizations. Once users are logged into the system, they are able to create their own graphs that allow them to visualize the data as wished by allowing the configuration of several parameters such as the resolution of the temporary data or the type of visualization.

2.6.5 Service implementation

2.6.5.1 Service implementation IT environment (dependencies like SQL db)

The Grafana baseline service needs to have data stored in a timeseries database. Since this is not directly provided by the SynchroniCity infrastructure, the Grafana administrator would need to prepare the environment by installing:

- QuantumLeap⁴⁹: an API developed in the scope of SmartSDK project. This component serves as a bridge to store OMA NGSI data to CrateDB,
- CrateDB⁵⁰, an open source real-time SQL database with integrated search that is simple to scale.
-

After installing and configuring those components, it will be necessary to make a subscription⁵¹ to Orion Context Broker on behalf of QuantumLeap in order to initialize the data flowing to CrateDB.

2.6.5.2 Service deployment dependencies (e.g. Orion Broker, oAuth server etc.)

The only service needed from the SynchroniCity infrastructure is the Orion Context Broker. This component will be the source of the data flow that will be finally stored in CrateDB as timeseries.

2.6.5.3 Installation and admin guide

The service is dockerized, general deployment files are provided and a deployment guide is already published on the public documentation⁵². A customized version of the deployment guide with the SynchroniCity framework will be written in future and reported for the next iteration of this deliverable.

⁴⁸ <https://smartsdk.github.io/ngsi-timeseries-api/>

⁴⁹ <https://smartsdk.github.io/ngsi-timeseries-api/>

⁵⁰ <https://crate.io/>

⁵¹ <https://smartsdk.github.io/ngsi-timeseries-api/user/#orion-subscription>

⁵² <http://hop-ubiquitous-documentation.readthedocs.io/en/latest/development.html#annex-6-deploy-and-use-of-grafana>

3 Conclusions and future developments

This first attempt of identification and implementation of common baseline services among Reference Zones is aiming at removing the traditional silo approach when deploying services in the urban ecosystems. Services are horizontally supported by partners and are going to be adopted by multiple cities and therefore scaling city platforms without vendor or city lock-in. New services can leverage those baseline services gaining time-to-market and inheriting architectural robustness and implementation quality.

The Functional Requirements identified within Task 3.1 and described in (SynchroniCity, D3.1, 2018) are covered in the SynchroniCity reference architecture (SynchroniCity, D2.1, 2017) and the baseline services depicted in this document. As shown in Table 32, already more the 70% of the FRs are addressed (see Annex for a full view).

	Common FRs	FRs Addressed	%
Human-centric traffic management	8	6	75
Multi-modal transportation	28	20	71.4
Community Policy Suite	8	5	62.5
	44	31	70.45

Table 32 Percentage of addressed FRs in common between RZs

This work shows the enactment of the SynchroniCity approach of re-using building blocks on top of common interfaces (the SynchroniCity northbound interfaces) for a shared effort towards implementation of city services. This establishes the technical foundation of a digital single market for IoT-enabled smart cities in Europe and beyond and therefore targeting the Project Objective (PO) 1 of the SynchroniCity project. In fact, the baseline services are part of the IoT ecosystem of solutions on top of the IoT infrastructure. In addition, the collaboration among partners paved the road for a co-creation approach of citizen services that are scalable and replicable which is addressing the Project Objective 3.

Some of the described baseline services in this document are at different maturity levels, for instance some needs to put into a docker, some other needs are still at a very early stage of design and need to be implemented. In the next months those services will advance in order to comply with the baseline services directives and guidelines given in (SynchroniCity, D3.4, 2018) and the final description of those services will find place into (SynchroniCity, D3.3, 2018).

In addition the work within the Task 3.3 for the definition of the use-cases and the scenarios of usage for each of the final applications will be the input for the identification of new baseline services and the creation of new developing teams that will lead to a shared effort and co-creation of new baseline services covering the rest of the FRs and enlarging the offer of IoT-enabled city services. New identified baseline services will be depicted in the next version of this document (SynchroniCity, D3.3, 2018).

Some new candidates to become SynchroniCity Baseline Services have been proposed as helpful for the implementation of the applications of RZs but they are still under study for assessing their applicability (e.g. constraints on availability of data or functional requirement matching).

A first candidate is the GTFS Realtime Service aims at providing GTFS-RT (GTFS-RealTime) updates, such as service alerts, trip updates and vehicle positions. Currently, real-time public transport trip predictions are available via the Routing Service. OpenTripPlanner accepts only one dynamic updater per feed, so a specific converter *siri2gtfsrt* inputs multiple data sources, such as service alerts and vehicle positions, and integrates them into GTFS-RT.

A second candidate is the Map Service providing background maps in Tile Map Service⁵³ format. Map data is based on vector export of OpenStreetMap. This service is already used in the Helsinki RZ. The maps are also annotated by various data points such as bus stops and park and ride locations.

Another candidate is the *Geocoding* service which provides mapping from addresses into geographic coordinates such as latitude and longitude in given coordinate system. A dual service, *Reverse Geocoding*, maps given coordinates into human readable addresses. When using map as an interface, results of geocoding can be visualized on a map with markers for a user. On the other hand, when a user clicks on a point on a map, the system translates that position into an address. Geocoding and Reverse Geocoding services use Pelias⁵⁴ that is a modular, open-source geocoder built on top of *Elasticsearch*⁵⁵ for fast and accurate global search. Pelias is completely open source and MIT licensed software. The importers filter, normalize, and ingest geographic datasets into the Pelias database. Currently there are five officially supported importers: *OpenStreetMap*, *OpenAddresses*, *Who's on First*, *Geonames* and *Polylines*⁵⁶.

⁵³ http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification

⁵⁴ <https://pelias.io/index.html>

⁵⁵ <https://www.elastic.co/>

⁵⁶ <https://digitransit.fi/en/developers/services/6-data-containers/geocoding-data/>

Bibliography

- DCAT-AP. (2015). *DCAT Application Profile for data portals in Europe*. EU.
- ETSI NGSI-LD. (2018). *ETSI GS CIM 004 - Draft*. ETSI.
- (4 May 2016). *General Data Protection Regulation - Official Journal of the European Union*, L 119.
- OMA. (2012). OMA NGSI-9/10. *NGSI Context Management*. OMA.
- SynchroniCity. (2017, August). D2.1. *Reference Architecture for IoT Enabled Smart Cities*. SynchroniCity.
- SynchroniCity. (2018, March). D2.2. *Guidelines for the definition of OASC Shared Data Models*. SynchroniCity.
- SynchroniCity. (2018). D3.1. *Specification and design of initial IoT applications*. SynchroniCity.
- SynchroniCity. (2018). D3.3. *Suite of baseline implementations - advanced*. SynchroniCity.
- SynchroniCity. (2018). D3.4. *Common methodology and toolset for city service customization*. SynchroniCity.
- SynchroniCity. (2018). D3.5. *Customized IoT service prototypes for lead ref. zones – basic*. SynchroniCity.
- SynchroniCity. (2019). D3.6. *Customized IoT service prototypes for lead ref. zones – advanced*. SynchroniCity.

Annex A: Functional Requirements addressed

		Avg	Addressed by
FR-8	The system should provide different views on the superimposed data sets	4,33	Smartcities Dashboard baseline service
FR-10	The system should allow selection of specified time window and display corresponding aggregated data over the selected time window	4,33	Metrics Visualizer (Grafana) baseline service
FR-11	The system may provide advice/prediction based on aggregated historical data	3,67	Metrics Visualizer (Grafana) baseline service
FR-6	The system must provide flexibility in terms of viewing, or having access to, certain data sets (e.g. through dashboard widgets)	3,33	Smartcities Dashboard baseline service; Metrics Visualizer (Grafana) baseline service
FR-5	The system must provide user identification and access control at least by a username/password basic security	3,33	SynchroniCity reference infrastructure
FR-7	The system may provide a map as the main way of interaction to superimpose data sets	3,00	Smartcities Dashboard baseline service
FR-12	The system may allow for annotation specified time windows	2,67	
FR-2	System should recognize preconfigured traffic situations and classify these situations in real-time	2,33	

Table 33 Human-Centric Traffic Management application theme FRs already addressed

		Avg	Addressed by
FR-4	The user should be able to enter (or provide by the GPS sensor embedded in his mobile device) the departure point, destination point and optionally other intermediates points of an itinerary on a map.	5,00	Routing baseline service
FR-5	The user should be able to enter the departure point, destination point and optionally other intermediates points of an itinerary writing their addresses, GPS coordinates or Point of Interest (Pol).	5,00	
FR-2	The application may provide basic features that can be accessed in an unregistered way.	4,75	
FR-6	The user should be able to select among different traveler profiles (e.g.: by walk, drive a car, drive a bike, use public transportation, etc.).	4,75	Routing baseline service

FR-8	The application must permit to plan routes by considering user preferences (panoramic itinerary, air quality, lowest cost, lowest time, etc.) and requirements	4,75	Routing baseline service
FR-9	The user should be able to view on map alternative routes and select the one he/she wants to use.	4,50	Routing baseline service
FR-11	The user should be able to select the preferred route he/she want to use.	4,25	
FR-13	The application may consider real-time information (deleted rides, delays, etc.) to better plan and update the journey.	4,25	
FR-14	The app should notify the user in real-time about issues that affect selected route.	4,25	
FR-17	The application must permit to search for car parks by considering user preferences (proximity of stations, bus stop, destination address, secured park, Park & Ride features, etc.)	4,00	
FR-29	Vehicle docking stations must at least provide User information, Vehicle info, Status and their GPS location.	4,00	SynchroniCity reference infrastructure
FR-32	Mobility premises must at least provide their GPS location and their status.	4,00	SynchroniCity reference infrastructure
FR-7	The user must be able to combine multiple transportation modes and mobility infrastructures (e.g.: metro + bike sharing for last mile instead of bus).	3,75	Routing baseline service
FR-16	The application must provide users with a georeferenced search of parking areas.	3,75	SynchroniCity reference infrastructure; Smartcities Dashboard baseline service
FR-18	The application must be able to indicate which transport possibilities are currently available near the selected parking area (e.g. estimated arrival times for nearby bus lines).	3,50	Smartcities Dashboard baseline service
FR-30	Vehicle sensors must at least provide Vehicle/transportation line info, Speed and their GPS location.	3,50	SynchroniCity reference infrastructure
FR-31	Environmental sensors must at least provide humidity, Air quality, noise and wind speed.	3,50	SynchroniCity reference infrastructure
FR-33	User devices must at least provide their GPS location.	3,50	
FR-1	The application must provide user identification	3,00	SynchroniCity

	and access control at least by a username/email address and password basic security (log-in).		reference infrastructure
FR-26	Traffic sensors must at least provide their GPS location and the traffic flow intensity.	3,00	Combining the SynchroniCity reference infrastructure and the Traffic Flow Estimator baseline service
FR-3	User must register an account and log-in to access features of the application that cannot be used in an unregistered way.	2,75	SynchroniCity reference infrastructure
FR-12	The user should be able to view on a map the points of interest located in an area near to the route.	2,75	Smartcities Dashboard baseline service
FR-15	The application should be able to consider user related information to suggest routes based on his/her preferences.	2,75	Routing baseline service
FR-19	The application should be able to indicate in real-time where free parking lots are available and at which cost/minute.	2,75	SynchroniCity reference infrastructure; Smartcities Dashboard baseline service
FR-20	The application may be able to estimate when and where it will be possible to find a free parking lot.	2,50	Parking estimation service
FR-21	The application may indicate alternative parking lots/areas nearby when there's no parking availability (or the parking availability estimated probability is too low) on the user-selected area.	2,25	
FR-10	The user should receive the amount/estimation of the cost (e.g. total ticket price) and arrival time (when possible) of each alternative route.	2,00	
FR-27	Parking sensors must at least provide their GPS location and their status.	2,00	SynchroniCity reference infrastructure

Table 34 Multi-Modal Transportation application theme FRs already addressed

		Avg	Addressed by
FR-1	Hierarchical login; system should cater for login and access to different dashboards / datasets based on the hierarchy of the local authority	4,67	Partially by SynchroniCity reference infrastructure
FR-3	Data management; system should be able to take custody of the data and edit for visualization	4,67	Partially by SynchroniCity reference infrastructure; Metrics Visualizer (Grafana) baseline service

FR-4	UI requirements; system should be intuitive and easy to use	4,67	Smartcities Dashboard baseline service; Metrics Visualizer (Grafana) baseline service
FR-5	User admin; authorities should be able to control their own teams / users via admin panels	3,67	
FR-2	Editable dashboards; based on city / team requirements user should be able to amend and lock (based on hierarchy) the dashboard "widgets"	3,33	Metrics Visualizer (Grafana) baseline service
FR-8	Meta reporting; for statistics on data and user interactions with the system	3,00	
FR-9	Workflows; for emailing up of reports and creating a level of autonomy between other systems	3,00	
FR-7	Connection to SynchroniCity digital marketplace; for north and south bound data purposes	2,67	

Table 35 Community Policy Suite application theme FRs already addressed