

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:	D3.3
Deliverable title:	Suite of baseline implementations - advanced
Document version:	Ver1.0
WP number:	WP3
Lead beneficiary:	35-NEC
Main author(s):	Flavio Cirillo, Detlef Straeten (NEC), Adrián Arroyo, David Gómez, Jose Gato (ATOS), Giuseppe Ciulla (ENG), Ignacio Elicegui, Luis Diez (UC), Jaime Ventura (POR), Antti Nurminen (FVH), Germán Molina Carrasco, Rubén Molina Moreno (HOPU)
Internal reviewers:	Luis Muñoz (UC), Reza Akhavan (FCC), Martin Brynskov (AU)
Type of deliverable:	Other
Dissemination level:	Public
Delivery date from Annex 1:	M22
Actual delivery date:	15.01.2020 (M37)



Co-funded by the
European Union

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



Switzerland



South Korea

Executive Summary

This document reports the outcome of the joint effort among SynchroniCity partners towards a common approach to smart cities solution. In particular the focus is on the isolated standalone components jointly developed and used as building blocks of the full-fledged Reference Zones (RZs) city services. The content of this report is to be considered, on the one hand, an advancement of deliverable D3.2 [1] since new features or even new atomic services have been implemented. On the other hand, the contents are complementary to the previous deliverable since unchanged architectures/interfaces/features are not reported here but referenced to such document.

It is important to highlight that there has been a change in convention regarding the name of the services treated in this document. Indeed, after the delivery of D3.2 what were known to be “**baseline services**” are now called “**atomic services**”. This was necessary since the previous name was too often motivation for confusion.

Target Audiences for this deliverable are exploiters of atomic 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 describe the advancements on features and/or development of the atomic services already identified in [1]
- To introduce the new atomic services identified since the publication of [1]
- To provide replicable reference implementations of these atomic services
- To define the way they should be installed, configured, distributed and deployed
- To settle down the foundations for future implementations meant to be performed in the remaining life of the SynchroniCity project and beyond.

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

- They are built on top of the SynchroniCity framework [2] and exploit the different interaction points offered by the platform, e.g. NGSI interfaces and APIs [3], historical data collection [4] and open data [5], Data Marketplace API¹ (that will be addressed in the future).
- Therefore, data input will be NGSI-compliant (in terms of data models [6]) so as to respect OASC²’s principles. Consequently, interoperability and replicability will be ensured.
- Alternatively, a SynchroniCity atomic service might be built on top of other atomic services and exploiting the output and the future of such atomic services.
- They provide 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 [7], and opening the door to new integration possibilities
- 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.tnforum.org/wiki/display/API/Open+API+Table>

² <http://oascities.org/>

The atomic services are a clear response to fulfil the macro-objective of SynchroniCity to increase the portability of IoT-enabled **city services** among cities, where cities services stand for applications built for citizens targeting citizens' needs with the aim of "smartify" the life within urban area. In fact, different RZs and respective technical partners have actively collaborated to the co-creation of those cities services, minimizing the efforts needed to develop them by distributing work for common features, resulting in atomic services. In addition, the effort spent can be re-used by third parties in order to boost the progress of an ecosystem. This enables replicability and demonstrates OASC principles and the Synchronicity framework advantages.

Evolving from the atomic services identification of D3.2, new features together with completely new atomic services have been identified. This is due to the advancement of the cities services in each RZs and therefore the identification of new requirements has happened. In addition some services are also reflecting the progresses of the SynchroniCity framework such as the definition of a data model for handling GTFS [8] and GTFS-RT [9] data through NGSI.

This deliverable is the final report on atomic services in SynchroniCity but the lifecycle of atomic services does not finish with this document. In fact, development of features and integration with other services is always advancing. In addition, it is foreseen a lifetime beyond SynchroniCity with the possibility of having the ecosystem of atomic services growing in features and components.

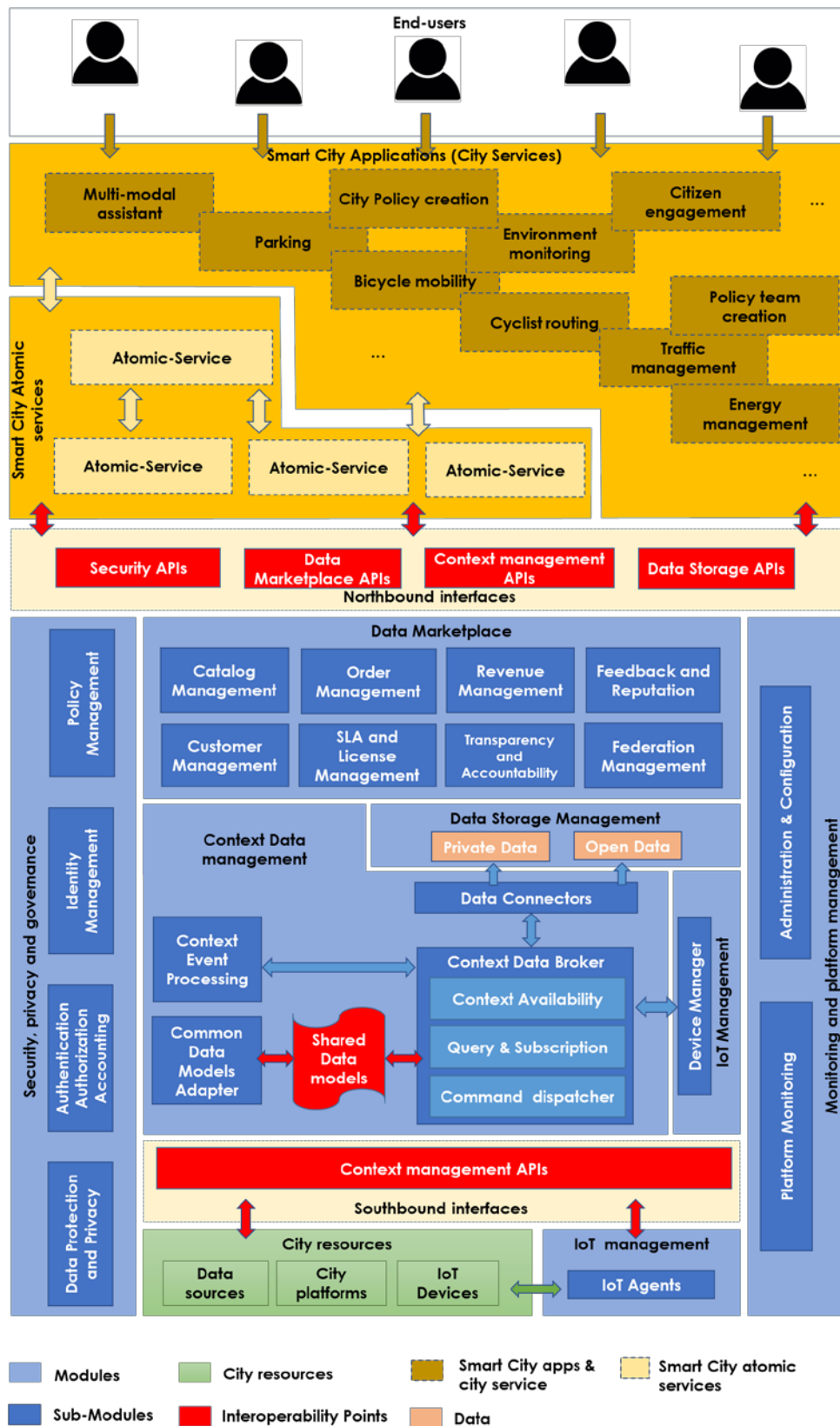


Figure 1 Full stack of the SynchroniCity Reference Architecture from sensors and data sources to end users (citizens). It is comprehensive of the atomic and city services layer on top of the SynchroniCity northbound interface

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CB	Context Broker
CSV	Comma Separated Values
D	Deliverable
EC	European Commission
ELK	Elastic Search, Logstash and Kibana
FR	Functional Requirement
GTFS	General Transit Feed Specification
GTFS-RT	GTFS- Realtime
IoT	Internet of Things
LGPL	Lesser General Public License,
LSTM	Long Short-Term Memory
NGSI	Next Generation Service Interface
OTP	Open Trip Planner
Pol	Point of Interest
REST	Representational State Transfer
RZ	Reference Zone
WP	Work Package
WT	Work Task

Contents

Contents

EXECUTIVE SUMMARY	2
ABBREVIATIONS.....	5
LIST OF FIGURES.....	9
LIST OF TABLES.....	10
1 INTRODUCTION.....	12
1.1 EVOLUTION OF ATOMIC SERVICE ROSTER.....	13
1.2 DISCLAIMER.....	14
2 ATOMIC SERVICES	15
2.1 ATOMIC SERVICES RESPONSIBILITIES AND ADOPTION MAP.....	15
2.1.1 RESPONSIBILITIES MAP	15
2.1.2 ADOPTION MAP	15
2.2 ROUTING ATOMIC SERVICE	17
2.2.1 DESCRIPTION.....	17
2.2.2 APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	17
2.2.3 API.....	18
2.2.4 SERVICE ARCHITECTURE	18
2.2.5 SERVICE DEPLOYMENT	18
2.2.6 SUPPORT	19
2.2.7 LICENSE & TERMS AND CONDITIONS	19
2.3 PARKING PROBABILITY ESTIMATOR ATOMIC SERVICE	20
2.3.1 DESCRIPTION.....	20
2.3.2 APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	20
2.3.3 INTERFACE	21
2.3.3.1 Parking Estimation API	21
2.3.3.1.1 Data Structures	24
2.3.3.2 Information Model	24
2.3.4 SERVICE ARCHITECTURE	24
2.3.5 SERVICE DEPLOYMENT	29
2.3.6 SUPPORT	31
2.3.7 LICENSE & TERMS AND CONDITIONS	31
2.4 TRAFFIC FLOW ESTIMATOR ATOMIC SERVICE	32
2.4.1 DESCRIPTION.....	32
2.4.2 APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	32
2.4.3 INTERFACE	33
2.4.3.1 Traffic Service API.....	33
2.4.3.2 Information Model	35
2.4.3.3 Interaction diagram (with the external world)	35

2.4.4	SERVICE ARCHITECTURE	35
2.4.5	SERVICE DEPLOYMENT	36
2.4.6	SUPPORT	36
2.4.7	LICENSE & TERMS AND CONDITIONS	36
2.5	SMARTCITIES DASHBOARD ATOMIC SERVICE	36
2.5.1	DESCRIPTION.....	37
2.5.2	APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	37
2.5.3	INTERFACE	38
2.5.4	SERVICE ARCHITECTURE	38
2.5.5	SERVICE DEPLOYMENT	38
2.5.6	SUPPORT	39
2.5.7	LICENSE & TERMS AND CONDITIONS	39
2.6	METRICS VISUALIZER (GRAFANA) ATOMIC SERVICE	40
2.6.1	DESCRIPTION.....	40
2.6.2	APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	41
2.6.3	INTERFACE	42
2.6.4	SERVICE ARCHITECTURE	42
2.6.4.1	Customization for Synchronicity	42
2.6.5	SERVICE DEPLOYMENT.....	46
2.6.6	SUPPORT	46
2.6.7	LICENSE & TERMS AND CONDITIONS	46
2.7	GTFS FETCHER ATOMIC SERVICE	47
2.7.1	DESCRIPTION.....	47
2.7.2	APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	47
2.7.3	API.....	47
2.7.4	SERVICE ARCHITECTURE	47
2.7.5	IMPLEMENTATION ROADMAP.....	48
2.7.6	SUPPORT	49
2.7.7	LICENSE & TERMS AND CONDITIONS	49
2.8	NGSI URBAN MOBILITY TO GTFS ATOMIC SERVICE	50
2.8.1	DESCRIPTION.....	50
2.8.2	APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	50
2.8.3	INTERFACE	51
2.8.3.1	Urban Mobility to GTFS API.....	51
2.8.3.2	Data Structures.....	54
2.8.3.3	Information Model and NGSI-GTFS mapping.....	59
2.8.3.4	Interaction Diagram	61
2.8.4	SERVICE ARCHITECTURE	65
2.8.4.1	Service Components.....	65
2.8.4.2	Non-functional requirements.....	67
2.8.5	IMPLEMENTATION ROADMAP/SERVICE DEPLOYMENT.....	67
2.8.5.1	Service implementation IT environment.....	67
2.8.5.2	Service deployment dependencies	67
2.8.5.3	Installation and configuration guide	68
2.8.6	SUPPORT	68
2.8.7	LICENSE & TERMS AND CONDITIONS	68
2.9	GTFS-RT LOADER FROM NGSI ATOMIC SERVICE	69
2.9.1	DESCRIPTION.....	69
2.9.2	APPLICATION THEME FUNCTIONAL REQUIREMENT ADDRESSED	69
2.9.3	INTERFACE	69
2.9.3.1	Management API.....	70
2.9.3.2	Service API	71

2.9.4 SERVICE ARCHITECTURE 71

2.9.5 IMPLEMENTATION ROADMAP/SERVICE DEPLOYMENT..... 72

2.9.6 SUPPORT 73

2.9.7 LICENSE & TERMS AND CONDITIONS 73

2.10 ATOMIC SERVICES SYNOPSIS 74

3 CONCLUSIONS AND FUTURE DEVELOPMENTS 76

BIBLIOGRAPHY 78

ANNEX A: FUNCTIONAL REQUIREMENTS ADDRESSED..... 82

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 atomic and city services layer on top of the SynchroniCity northbound interface	4
Figure 2 The web interface of the generic OTP service	17
Figure 3 Parking probability estimator service architecture and interaction with the SynchroniCity framework.....	25
Figure 4: Data flow over AI Prediction Engine, example configuration.....	27
Figure 5: Long Short-Term Layer internal structures [25]	28
Figure 6: Dense feed forward internal layers (example) [26]	28
Figure 7 Traffic Flow estimator service architecture and interaction with the SynchroniCity framework	36
Figure 8 SmartCities Dashboard view.....	37
Figure 9 Grafana Dashboard example	40
Figure 10 Grafana Worldmap customization for SynchroniCity.....	42
Figure 11 Configuration of the Worldmap plugin	43
Figure 12 Metrics Visualizer (Grafana) – Worldmap plugin: Representation Options	44
Figure 13 Worldmap "onclick" option.....	44
Figure 14 Worldmap tile options.....	44
Figure 15 Customized plugin showing parking items	45
Figure 16 Customized plugin showing beaches items	46
Figure 17 – GTFS Fetcher Service architecture and basic running environment.....	48
Figure 18 - GTFS data model diagram.....	59
Figure 19 - getAgencies Sequence Diagram.....	62
Figure 20 - getRoutes Sequence Diagram.....	62
Figure 21 - getTrips Sequence Diagram	63
Figure 22 - getGTFSByAgency Sequence Diagram.....	64
Figure 23 - GTFSFeed update job	65
Figure 24 - Urban Mobility to GTFS Service architecture	66
Figure 25 GTFS-RT loader from NGSI service architecture.....	72

List of Tables

Table 1 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.....	15
Table 2 Adoption map between services and applications developed by RZs.....	15
Table 3 Multi-modal transportation application theme FRs addressed.....	18
Table 4 Multi-modal transportation application theme FRs addressed.....	21
Table 5 List Parking Areas API	22
Table 6 Parking Area Detail API.....	22
Table 7 Free Parking Spot Estimation API.....	23
Table 8 Parking estimator output data model.....	24
Table 9 Multi-modal transportation application theme FRs addressed.....	33
Table 10 List Traffic Flows API.....	34
Table 11 Current Traffic Flow Conditions API	34
Table 12 Traffic Flow Conditions Estimation API	35
Table 13. FRs of the Human-Centric Traffic Management application theme addressed by the Smartcities Dashboard atomic service	38
Table 14 FRs of the Multi-Modal Transportation application theme addressed by the Smartcities Dashboard atomic service	38
Table 15 FRs of the Community Policy Suite application theme addressed by the Smartcities Dashboard atomic service	38
Table 16 FRs of the Human-Centric Traffic Management application theme addressed by the Metric Visualizer (Grafana) atomic service	41
Table 17 FRs of the Community Policy Suite application theme addressed by the Metric Visualizer (Grafana) atomic service	41
Table 18 Worldmap plugin PopUps configuration	45
Table 19 Multi-modal transportation application theme FRs addressed.....	47
Table 20 Multi-modal transportation application theme FRs addressed.....	51
Table 21 - Get Agencies API.....	52
Table 22 - Get Routes API	53
Table 23 - Get Trips API.....	53
Table 24 - Get GTFS By Agency.....	54
Table 25 gtfs:Agency entity	55
Table 26 gtfs:Route entity	56
Table 27 gtfs:Trip entity.....	57
Table 28 – ErrorResponse entity.....	58
Table 29 GTFS Feed Entity and contained files.....	59
Table 30 - GTFS members - NGSI entities mapping.....	60
Table 31 - Agency mapping	60
Table 32 - Route mapping.....	60
Table 33 - Trip mapping	61
Table 34 - Route relationship fields.....	61

Table 35 - Trip relationship fields	61
Table 36 Multi-modal transportation application theme FRs addressed.....	69
Table 37 Add GTFS feed method	70
Table 38 List GTFS feeds method	71
Table 39 Get GTFS-RT feed method	71
Table 40 Summary of the important links of the atomic services suite.....	75
Table 41 Percentage of addressed FRs in common between RZs	76
Table 42 Human-Centric Traffic Management application theme FRs already addressed	82
Table 43 Multi-Modal Transportation application theme FRs already addressed.....	84
Table 44 Community Policy Suite application theme FRs already addressed.....	85

1 Introduction

This deliverable presents the final result of Task 3.2 which targets the identification and implementation of atomic services that are used as building blocks for the implementation of applications and city services. The content of this report is to be considered, on the one hand, an advancement of D3.2 [1] since new features or even new atomic services have been implemented. On the other hand, the contents are complementary to the previous deliverable since unchanged architectures/interfaces/features are not reported here but referenced to such document.

Since the delivery of [1] there was the necessity to changing the name of the formerly known as “baseline services” to “atomic services”. This change in terminology has been made in order to avoid confusion on the nature of the services. In fact, the atomic services are placed between the SynchroniCity base framework [10] and the RZs application (i.e. cities services), as shown in Figure 1. They act as standalone components on top of the base framework to be used as building blocks for the cities services. Atomic services can be used as a standalone or in composition for achieving more advanced backend service for the final city service. The nature of the atomic 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 atomic services are delivered in a way that are easily customized by following the methodology and using the tools presented in the deliverable D3.4 [11].

The pilot SynchroniCity’s city services are categorized in three application themes [12]:

- *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 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 atomic 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 atomic services. All atomic services will be at the end wrapped into a docker container following the directives of [11].

As outlined in [1] for the identification of atomic services we have followed 2 approaches:

1. The bottom-up approach that foresees the shaping of the architecture of RZs application starting from useful components already available (that was the case for the Routing atomic service, the Smartcities Dashboard service and the Metrics Visualizer services)
2. The top-down approach that suggested new components by analyzing RZs applications requirements and architectures and identifying common challenges to be addressed with the same features. This approach brought to the definition of the Parking Probability Estimator service and the Traffic Flow Estimator services already at the time of [1], together with the new “GTFS Fetcher”, “NGSI Urban Mobility to GTFS” and “GTFS-RT Loader from NGSI” atomic services.

We foresee that the evolution of atomic services roster (Section 1.1) and functionalities might change in the future due, on the one hand, to the progresses of the RZs services, and, on the other hand, to the implementation of applications by third parties (such as Open Calls winners and future SynchroniCity participants beyond the official EU project end).

Last, but not least, it is worth bringing up the fact of how these atomic services are thought to be exposed and shared. On the one hand, at the time of writing this document, we count on two different "access points" where either internal or external developers can have (open) access to and play around with the out-of-the-box atomic services: GitLab [13] where all the source code is available, and people can not only deploy our solutions on their own infrastructures but also contribute to it, e.g. fork the latest version of a component, streamline it and then call for a merge request. Aside the source code itself, we have prepared a fully-fledged Docker image repository on Docker Hub <REF to Docker Hub>, where all the atomic services have an image ready to be deployed in a straightforward manner. Moreover, we must state that we have followed a set of best practices when it comes to properly document everything there, where we not only focus on the description of the atomic services, but thoroughly explain how to adequately install, configure and run them.

During the last review, it was remarked that it would have been great to have a kind of marketplace to make atomic services available. But it was not conceived from the very beginning and now seemed hard to introduce a third option. Even though Github and DockerHub repositories could be considered as a kind of marketplace (at least from a developer point of view), they lack some functionalities, such as trading and purchasing. In any case, atomic services packaging, documenting and technologies are considered for an easy landing in any kind of Marketplace. In particular the packaging into docker images that can be easily distributed and uploaded to any marketplace. The SynchroniCity consortium has in its roadmap the possibility of sharing/trading docker images, so ecosystem transaction management would be extended to support this possibility. In that case, it would be a great opportunity to test it as a possible marketplace, and atomic services will be ready for that.

1.1 Evolution of Atomic Service Roster

The process of selecting atomic services, as described in [1] (i.e. top-down and bottom-up approaches), has been demonstrated effective since it has speeded up the development of cities services and boosted the collaboration among RZs. A clear example of the bottom-up approach effectiveness has been the adoption of the Routing atomic service. It has been confirmed by many partners that this component is bringing many functionalities which have greatly eased the development of cities services. Currently four cities (Helsinki, Santander, Porto and Milan) are using it as an important component of their applications. The top-down approach has been likewise successful with the definition and developments of the two estimator atomic services (Parking Probability Estimator and Traffic Flow Estimator). In fact, those two atomic services were identified at time of a specific Santander application architecture and the features were of interest to other RZs: Milan and Carouge for the Parking Probability Estimator and Porto for the Traffic Flow Estimator. Other three atomic services (related to GTFS data) are instead an outcome of a hybrid approach. That is the necessity of integrating a given atomic service useful to many RZs (i.e. the Routing atomic service) with data coming from the SynchroniCity infrastructure.

All the initial roster of atomic services has been confirmed due to the interest and the progressively adoption by more and more RZs. This is the case for the Routing and the two estimation atomic services (for parking and traffic flow). A different consideration needs to be done for the two visualization services (the Smartcities Dashboard and the Metrics Visualizer atomic services). While it is true that only one found applicability in one RZ (the Metrics Visualizer in Eindhoven), it is also true that both eased the prototyping of applications on top of SynchroniCity. As an example, the Metrics Visualizer has been used for live demonstration of the atomic services concepts, in particular for visualizing the Parking Probability Estimator atomic services output.

The methodology of [11] for atomic services has been demonstrated as appropriate for facilitating the adoption of atomic services by RZs. Besides the good mix of high-level and technical description

(from concept to APIs), the adoption of docker containers helped the integration of the atomic services as off-the-shelf black box in the whole RZs application.

1.2 Disclaimer

The atomic 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 atomic services are under the full control of the atomic services' responsible teams and they are in charge of checking any license and restrictions of third parties' tools and/or services. Moreover, atomic services are delivered with different licenses and terms and conditions, therefore we invite the interested users to check those before their usage.

2 Atomic services

2.1 Atomic Services responsibilities and adoption map

2.1.1 Responsibilities Map

	Routing service	Parking Probability Estimator service	Traffic Flow Estimator service	Smartcities Dashboard service	Metrics Visualizer (Grafana) service	GTFS Fetcher service	NGSI Urban Mobility to GTFS	GTFS-RT loader from NGSI
ATOS								
ENG								
FVH								
HOPU								
Milan								
Porto								
TST								
UniCAN								

Table 1 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 1 shows the responsible teams within the consortium for each service. The responsible team is the one who 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 a third party. In any case the responsible team is the one who will be firstly contacted for supporting atomic 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 future due to convergence of interests or any other reasons.

2.1.2 Adoption Map

		Routing service	Parking Probability Estimator service	Traffic Flow Estimator service	Smartcities Dashboard service	Metrics Visualizer (Grafana) service	GTFS Fetcher service	NGSI Urban Mobility to GTFS	GTFS-RT loader from NGSI
Human- centric traffic mgmt	Ant								
	Ein								
	Mil								
Multi-modal transportation	Hel								
	Mil								
	Por								
	San								
Community Policy Suite	Car								
	Man								
	Por								

Table 2 Adoption map between services and applications developed by RZs

Table 2 presents the first adoption map between applications developed by RZs and the current list of services. The Routing Service is already 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 atomic services; the reader is invited to consult the outcomes

of Task 3.3 for more details [14]. In addition, in spite of being used by no applications (such the case of the Metrics Visualizer (Grafana) atomic service), some atomic services are still considered part of the SynchroniCity portfolio in order to have wider offer for third parties, fast prototyping, and for future exploitation of RZs.

2.2 Routing atomic service

The Routing atomic service was subject of no changes since [1] was released. Considering its powerful features and importance for many RZs, a big effort has been made in order to integrate it with the SynchroniCity framework, resulting in three new atomic services to be used in combination with the Routing atomic service: GTFS Fetcher (see Section §2.7), NGSI Urban Mobility to GTFS (see Section §2.8) and GTFS-RT Loader from NGSI (see Section §2.9) atomic services.

2.2.1 Description

The Routing atomic service allows users to execute queries for finding routes, bus and taxi stops, city bikes and bicycling routes, and disruption info. In addition, it is useful for generating multi-modal routing, taking into consideration static and dynamic information coming from the cities such as air quality or traffic estimation.

This atomic service is based on the OpenTripPlanner (OTP)³. Requests to this service can be done either through the original OTP API [15] or via GraphQL [16]. The latter is a specific customization of the general service.

City information can be injected via the SynchroniCity framework with the usage of other SynchroniCity atomic services such as GTFS Fetcher, GTFS-RT Loader from NGSI, and Urban Mobility to GTFS.

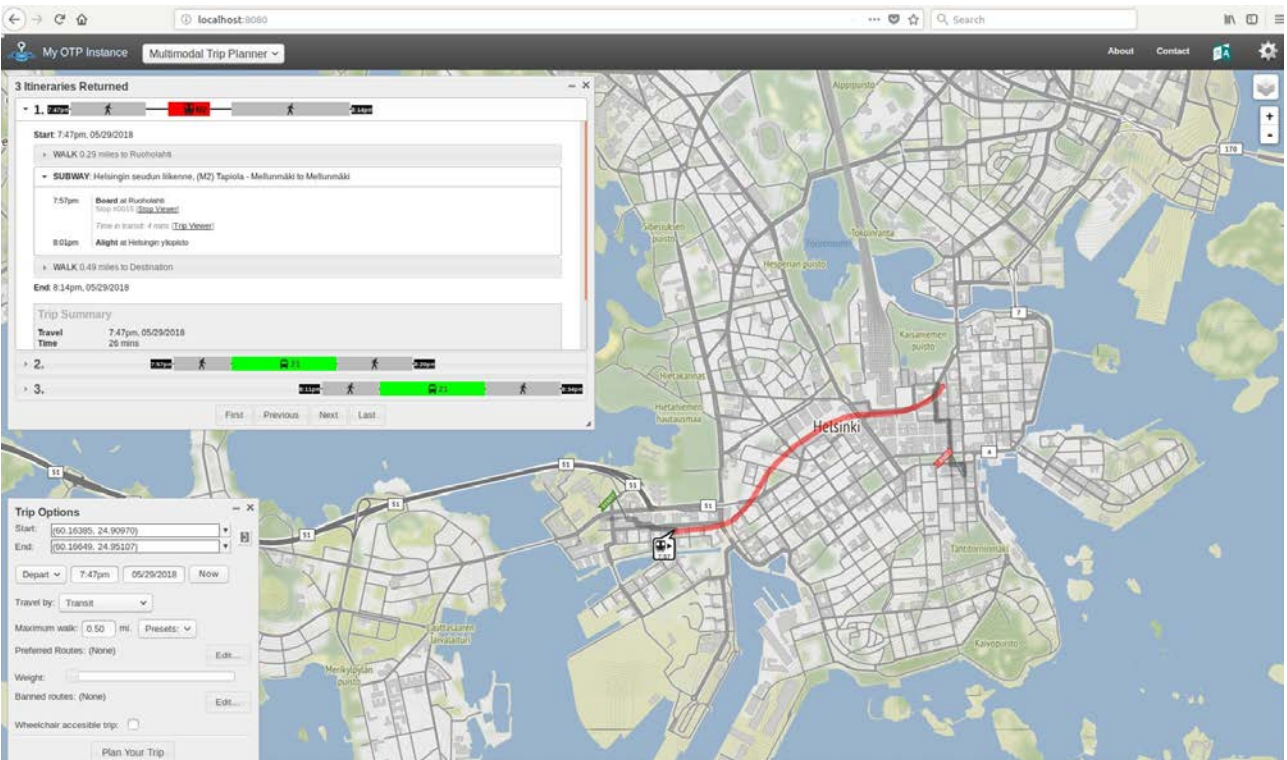


Figure 2 The web interface of the generic OTP service

2.2.2 Application theme Functional Requirement addressed

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

³ <http://www.opentripplanner.org>

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 routing 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 (PoI).	Accepted as input by the routing service in cooperation with the “GTFS fetcher”, NGSI Urban Mobility to GTFS” and GTFS-RT Loader” atomic services
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 and supported by the GTFS-RT Loader atomic 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
FR-13	The application may consider real-time information (deleted rides, delays, etc.) to better plan and update the journey.	The routing services accept real-time information for elaborating a route. The input real-time data is injected through the “GTFS-RT Loader from NGSI” atomic services
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 3 Multi-modal transportation application theme FRs addressed

2.2.3 API

The interfaces used by this atomic service are:

- The original REST API provided by OpenTripPlanner, and
- GraphQL interface customized by the Digitransit application⁴.

Since there are no changes in the interface since [1] we invite the interested reader to consult that document.

2.2.4 Service Architecture

Also in the service architecture there are no changes since [1]

2.2.5 Service Deployment

The provision of the service has been already depicted in [1] therefore we suggest the interested user to refer to this document.

⁴ <https://digitransit.fi/en/developers/architecture/x-apis/1-routing-api/>

2.2.6 Support

The Routing Atomic Service, OpenTripPlanner, is being developed by the OTP community. The community aims at stable software, which can be used by large entities, i.e. cities. Documentation is available at the development repository⁵ and at the *Readthedocs* site⁶. There is also a Google Group *OpenTripPlanner Developers*⁷ where one can find various questions and answers by the developers. Helsinki Region Transport, one of the largest users of OTP, has set up their own developer community with documentation for their Digitransit Open Source Platform, which uses their variation of OTP⁸.

The SynchroniCity Routing service has also its own support page dedicated to the SynchroniCity community⁹.

2.2.7 License & Terms and Conditions

OpenTripPlanner is distributed and developed under the GNU Lesser General Public License (LGPL), version 3 or any later version. LGPL is a free-software license published by the Free Software Foundation. The license allows developers and companies to use and integrate software released under the LGPL into their own (even proprietary) software without being required by the terms of a strong copyleft license to release the source code of their own components. The license only requires software under the LGPL be modifiable by end users via source-code availability. For proprietary software, code under the LGPL is usually used in the form of a shared library, so that there is a clear separation between the proprietary and LGPL components. The LGPL is primarily used for software libraries, although it is also used by some stand-alone applications.

OTP's License is available at the development repository¹⁰.

⁵ <https://github.com/opentripplanner/>

⁶ <https://opentripplanner.readthedocs.io/en/latest/>

⁷ <https://groups.google.com/forum/#!forum/opentripplanner-dev>

⁸ <https://dev.hsl.fi/>

⁹ <https://gitlab.com/synchronicity-iot/routing-atomic-service/issues>

¹⁰ <https://github.com/opentripplanner/OpenTripPlanner/blob/master/LICENSE>

2.3 Parking Probability Estimator atomic service

The content covered by this section reflects all the modifications and new features that were not present at the first iteration of this deliverable [17]. Should the reader need some complementary material, we refer to the following sources (they have a living nature and will be properly updated as long as new features appear).

- Parking estimator API module - Project's GitLab [18]
- Parking estimator API documentation – Apiary [19]
- Parking estimator – Docker Hub [20]
- AI prediction module – Docker Hub [21]

In particular, whereas these atomic service data structures were not subject of changes, some methods and information model have been added/modified in the API. More details are given in Section 2.3.3.

The implemented architecture of this atomic service follows a micro-service approach, as it is detailed in Section §2.3.4.

2.3.1 Description

By means of Artificial Intelligence (AI) techniques, this atomic service brings the possibility of predicting the probability of finding a free parking spot within a particular area of a RZ. In terms of the time window of the estimation, we offer up to one-hour time ahead, split into four different discrete values: 15, 30, 45 and 60 minutes.

To achieve this, we cater a fully-fledge AI prediction module (which is explained in depth in Section §2.3.4) with:

- Historical information
- (Near) real time data received via subscription mechanisms

That is, at the time of booting the service (say, $t=0$), the whole history of the various context elements is imported/dumped onto the prediction module; after that, upon any new observation retrieved from any of the devices, the parking estimator API automatically forwards it to the same sink. This way we keep the system updated to the most recent events happened in the involved cities.

Furthermore, in order to make the service more valuable, we feed the model with multiple features, as the ones listed below:

- **Parking area ratio** (that is, the ratio between free and occupied parking spots in a particular area). As a side node, this is not a parameter explicitly exposed in the context element payload, but it has to be calculated (trivial operation, though).
- **Type of vehicle**. As can be easily inferred, there exists a certain correlation between the availability of parking spaces and the category of the vehicle (e.g. car, adapted vehicle for people with any disability, etc.).
- **Working hours**. As the monitored areas in the different testbeds focus on critical zones in the cities (e.g. downtown), it can be also guessed that there exists a behavioral pattern that differentiate between, for instance, working-office hours (where people leave the car for many hours) and leisure hours (where the average stay time is shorter).

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 4 Multi-modal transportation application theme FRs addressed

2.3.3 Interface

2.3.3.1 Parking Estimation API

From the API specification presented in [1], we highlight in this section those methods that have been subject of change, leaving the others intact. Nonetheless, a complete and updated specification of the API can be found in [19].

- **List Parking Areas:** though the service main goal remains almost alike from the legacy version, we have slightly revamped the way it handles the pagination, thus making it more intuitive. It is outlined in Table 5. As a note, if the query parameters about pagination are not specified, all the context elements are shown in a single-page mode.
- **Parking Area Detail:** No changes from the legacy version. See Table 6.
- **Free Parking Spots Estimation:** as for the prediction, the first version allowed the specification of the time within which we aimed the estimation (that is, a discrete value that could be 15, 30, 45 or 60). However, after testing the API, we felt that this was not very user friendly when it came to integrate with e.g. a dashboard (we need to make 4 calls if we want to get the next hour estimation). Therefore, we have removed this parameter and made the method simple. Now, it looks as shown in Table 7.

API Name		List Parking Areas			
Description		This service will return the paginated list of the registered Parking Areas (OnStreetParking or OffStreetParking).			
End-Point		/parkingareas{?page,items_per_page,type}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
page	Query	number	No	1	Page number
items_per_page	Query	number	No	-	Number of context elements displayed per page. If nothing is specified, all the elements are displayed
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 Code	Status	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 5 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 6 Parking Area Detail API

API Name		Free Parking Spot 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. Unlike the previous version of the API, we have ditched the time window parameter, and the service returns in a single request the values for the next hour (i.e. 15, 30, 45 and 60 minutes ahead the current timestamp)			
End-Point		/parkingareas/{parkingareaID}/freespotsEstimation{?category}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
parkingareaID	Path	String	Yes	-	The unique identifier of the parking area.
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 (freeSpotEstimation) 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 7 Free Parking Spot Estimation API

2.3.3.1.1 Data Structures

As the data structures have been subjected to no changes since [1] we invite the interested readers to consult that document or the published API [19].

2.3.3.2 Information Model

Since the information model has been subjected to no changes since [1] we invite the interested readers to consult that document or the published API [18]. A missing data model was missing from the previous documentation and it is introduced below.

Embedded within OnStreetParking/OffStreetParking context information, the estimator returns a new field with the information of the predicted next hour. As a matter of fact, we also included some metadata that helps interpret the output. Namely, the “freeSpotEstimation” attribute contains of an array of 5 elements, where the first one shows the last observation “seen” by the context element, whereas the next four span the prediction of the next hour (showing the values for the next 15, 30, 45 and 60 minutes, respectively). Table 8 shows how the new attribute looks like. It is worth mentioning here that we may include more attribute to the metadata in order to add more information about the service (e.g. estimation recall, latency, etc.).

Estimator output		
Attribute	Value	Notes
freeSpotEstimation	[Last ratio, next 15, next 30, next 45, next 60]	
estimationTimes (part of freeSpotEstimation's metadata field)	[Last timestamp, next 15, next 30, next 45, next 60]	The metadata field contains an attribute called estimationTimes, which embraces the timestamps of the corresponding predictions held in the freeSpotEstimation attribute

Table 8 Parking estimator output data model

2.3.4 Service Architecture

We have split the “monolithical” service of the previous version of this atomic service into several micro-services, where each of them plays a dedicated role that, among all, shape the parking estimator atomic service.

Figure 3 illustrates how the services interact with each other and how we get the information from the SynchroniCity framework (bottom part of the figure). It is worth highlighting that all the details provided in this document complement what were written in the first iteration of this deliverable.

As depicted in Figure 3, the service relies on SynchroniCity architecture and defined information and data models [6] to capture the required information. It will also provide a RESTful standard-based API to request the outcomes.

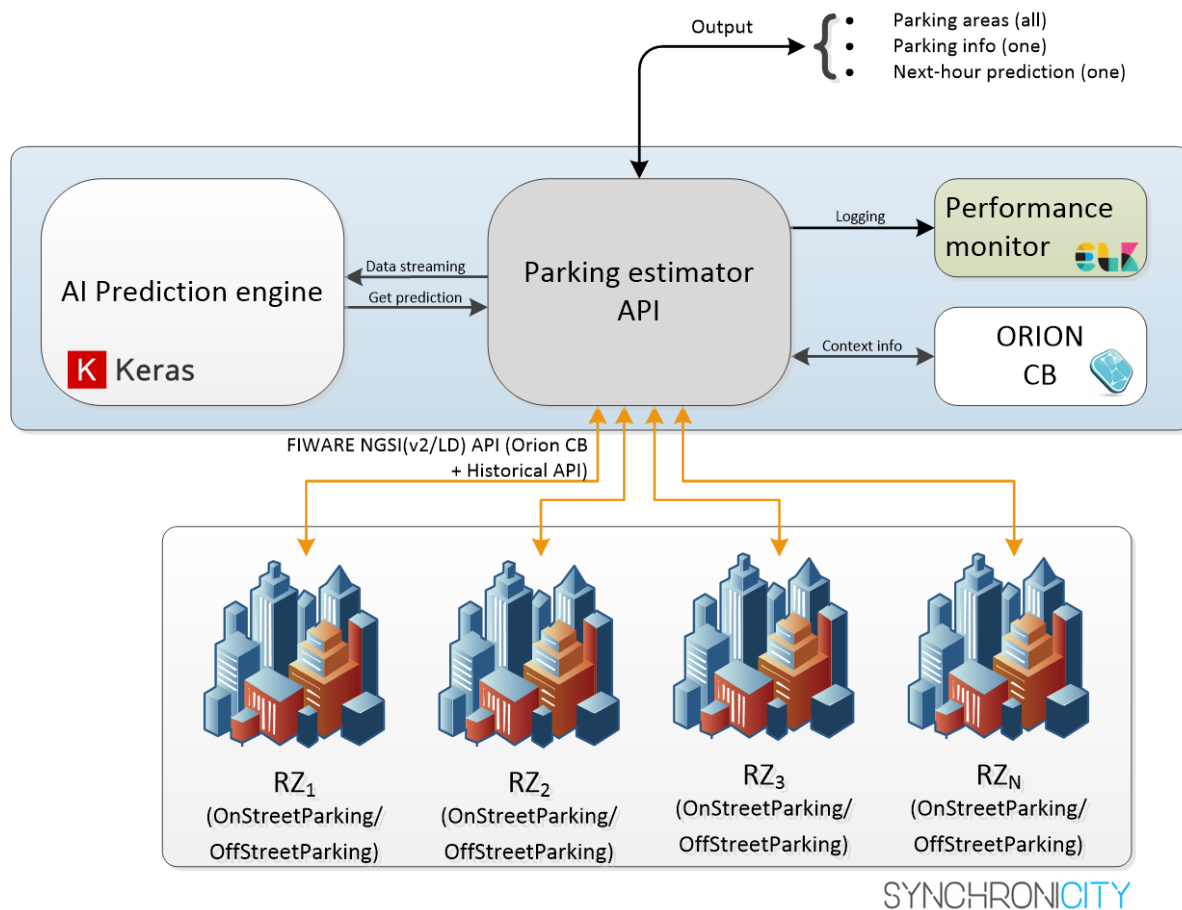


Figure 3 Parking probability estimator service architecture and interaction with the SynchroniCity framework

The parking estimator atomic service (framed within the light-blue box) is made of four different micro-services, connected one to each other via REST-based interfaces. Apart from the atomic service itself, we can observe at the lower part of the figure the underlying SynchroniCity frameworks, belonging to the different RZs. As can be easily inferred, the cities will cater the estimator with information on their respecting parking areas. Finally, on top of the atomic service, we have carried out a brand new RESTful API with three different methods/services:

- 1) get the list of parking areas;
- 2) get the information of a particular parking area;
- 3) receive the estimation of the next hour (at a particular parking area).

As for the atomic service per se, below we outline the main functionalities covered by the different micro-services: Parking Estimator API, AI Prediction Engine, Performance Monitor, and Orion Context Broker.

Parking estimator API:

This is the information broker that receives/processes/sends information from/to the other component, or the outside world. Namely, these are the connections and the features covered in all the cases:

- **RZs.** the Parking estimator API gathers the information from all the RZs (i.e. those ones that own OnStreetParking/OffStreetParking entities. As for the interplay with them, we distinguish between two different phases:
 - Clean-slate state (i.e. $t=0$), where the AI prediction module has not received anything

yet. At this initial phase, the Parking Estimator API queries, via

- o Subscription-based updates. After the first stage described in the previous point, the module sticks to every kind of event produced in any of the “discovered” context elements. This way, upon every notification, the new data is then forwarded to the AI prediction module, which, as will be explained later, will periodically “re-train” the models.
- **AI Prediction Engine.** After the initial dump, where the historical data from the different context elements are delivered to this module, this service proceeds to train the models (one per each context element). When this transitory has passed, the Parking Estimator API caters this module, as hinted before.
- **Orion CB.** Due to the default lack of rights to write on the different RZs’ Context Brokers, the atomic service holds its own Context Broker, where it keeps the information of the different context elements. Besides, it also has a twofold role, since it is also used as a kind of cache, thus avoiding asking to the AI prediction module upon any client sends a request. This way we achieve to significantly reduce the computational overhead (subsequent queries on the same context element would return the same output, hence it would be a waste of time, energy and computational resources).
- **Performance monitor.** In order to have a centralized logging sink, every “loggable” message will be “dumped” onto this module. For instance, the time taken to yield a prediction, the IP address of a client requesting info of a parking area, etc. We will harness all this information to carry out a thorough analysis of the whole system, thus leading us the opportunity to e.g. detect gaps and come up with solutions to improve the performance.
- **External clients.** This is the real output of the atomic service, the connection to the outside world. As defined in Section §2.3.3.1, we offer three different methods, where users can retrieve the whole list of OnStreetParking/OffStreetParking context elements, get the information of one of them or, as the most valuable feature of this atomic service, obtain the estimation (in terms of probability) for the next hour.

AI Prediction Engine:

This module corresponds to the core of the estimation operations. It deals with the typical workflow of most AI applications: data processing, learning/training and predicting/estimating, but is defined as an independent server with a REST API that offers an easy communication with the Parking Estimator API.

Since the instructions about how the module is used are available online in the project's Docker-Hub¹¹, here it will be described mainly how it works internally.

¹¹ <https://cloud.docker.com/u/synchronicityiot/repository/docker/synchronicityiot/ai-prediction-engine>



Figure 4: Data flow over AI Prediction Engine, example configuration

Figure 4 describes an example of how the data is treated over the module; in which the input is a streaming of live data, and the output is the estimation of the probability of finding a free parking spot. How the data is formed is subject to changes (1...N features), as described in the configuration of the module in the repository's documentation. The steps through the workflow are described as follows:

1. Data is continuously received from the Parking Estimator API, merged from all the entities connected. This data is stored so that, when there is enough of it, trainings of the estimation models can be made. This training operation is done periodically (period is configurable) to update these models according to the current behavior of the parking data. However, it is not done continuously as new data arrives, in order to avoid overloading the system with lots of small training operations that might not result in any improvement. It is important to mention that this is not a state-of-the-art online learning approach, but rather an adjustment of the models with more data. Another important fact is that every data related to the parking must be linked with a timestamp, as shown on the top right corner of Figure 4.
2. In order to train the system, the data stored is rephrased as a time series, which means that, for every piece of data (timestamp and features) an array of configurable size is built, in which every position corresponds to a previous piece of data (features corresponding to a previous timestamp). Between two consecutive positions, there is an interval of time (configurable) which is the same for every column of the array.
3. The last value in the array corresponds to the label (the value that we would like to estimate). The rest of the data in the series is used as the input training data for the models. How these models' architecture are built, is described later in this section.
4. Once the model is trained, it is deployed to attend to estimation requests. Similar to the training data processing, a unique time series array is built when a prediction request is made, which has the same structure that would have for the training phase, but this array is used as input for the prediction model. The output is sent to the Parking Estimator API, which will handle it and send to the other modules of the system.

Even though different machine learning techniques are used along the learning process, the core of the module makes us of Keras [22] high-level API, with Google's Tensorflow [23] deep learning library as back-end. Currently, two model architectures are being tested:

1. Model based on *LSTMs (Long Short-Term Memory [24]) layers*. This class of layer belongs to the recurrent neural network kind, where the output of a neuron is passed to a successor, which in this case is the same neuron. Figure 5 shows an unrolled recurrent layer example.

These deep learning networks have been proven to work efficient and accurately over time series problems (looking at Figure 5 it appears to be a series). Nonetheless, they require more time to train and a lot of data.

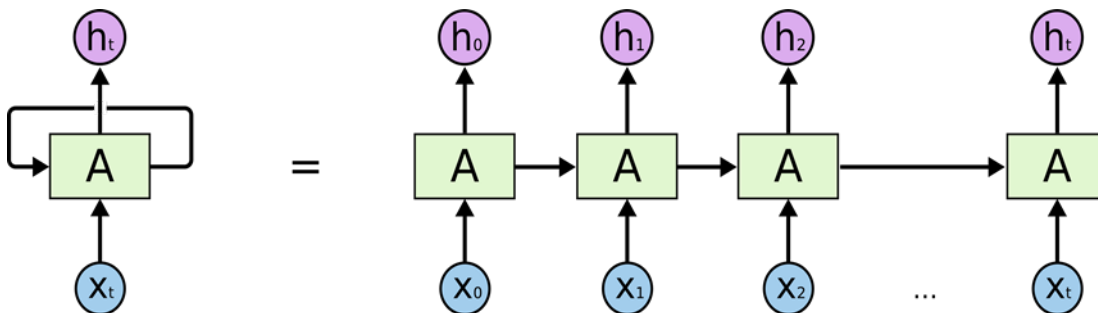


Figure 5: Long Short-Term Layer internal structures [25]

2. Model based on *Dense feed forward layers*. This is a typical deep learning network in which the first layer has the same inputs as the positions in the time series. They require less data and less training time than LSTMs, but longer sequences; which hinder the accuracy enhancement. An example of a small neural network based on this architecture is shown in Figure 6.

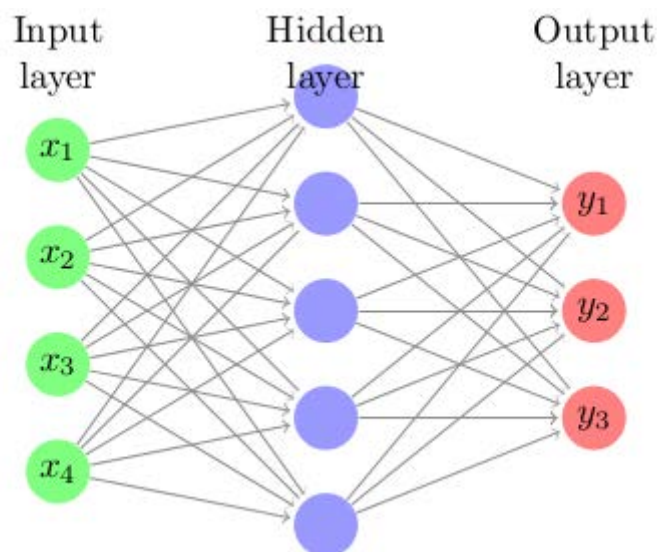


Figure 6: Dense feed forward internal layers (example) [26]

The goal of the system during training is minimizing the mean squared error of the predictions over the real values. This is carried out by continuous fits of the models over new data.

Performance monitor

It is based on an ELK (Elastic Search, Logstash and Kibana) stack¹², all the events that happen within the atomic service are recorded in this module. They are subsequently analyzed, in order to, for instance, spot flaws and assess the system performance. Nonetheless, it is worth mentioning that this component is optional and might not be part of the final deployment (see Section §2.3.5).

Orion Context Broker

¹² <https://www.elastic.co/elk-stack>

Like the RZs, the parking estimator atomic service also contains an Orion instance¹³, due to the lack of permission to update the context elements at the RZs (this is likely to happen, as was explained before). Thanks to this component (which works together with e.g. a MongoDB database), we can persist the information about the different context elements and reduce the number of queries addressed to either the RZs or the AI prediction module.

2.3.5 Service Deployment

Even though it is feasible to reproduce and run the atomic service by downloading and executing it (i.e. the respective components) from the source code (GitLab), we encourage to leverage the features and facilities offered by Docker and docker-compose for portability and re-usability, as we highlighted in [11].

As was shown in Figure 3, this atomic service works as the combination of various micro-services where, apart from the “central” Parking estimator API, we also need an AI prediction module, an Orion Context Broker instance and an ELK stack. The latest two modules are not vital for the services to work but are helpful, for instance, for minimizing queries to the SynchroniCity framework or assess the system performances.

In fact, we focus in this section on the deployment on the main two components, leaving the optional features out of the document. Nonetheless, the reader might refer to the Parking Estimator API Docker Hub [20] in order to get the most updated instructions of how to properly deploy the whole atomic service.

Before starting with the step-by-step guide, users **MUST** have installed the Docker framework on their respective devices (as a side note, we have tested the service on a Raspberry Pi and it still works; the performance is quite poor, though). Besides, for the sake of straightforwardness to deploy, we also encourage to install Docker Compose¹⁴ too. Saying it in layman’s terms, quoted from their documentation: “It is a tool for defining and running multi-container Docker applications”. Or saying it in other words, it will hide all the complexity of e.g. Docker command-line setup behind a single configuration file (indeed, we do provide it).

Thanks to Docker, all the underlying technologies/frameworks are abstracted and nothing else is required when it comes to prepare the system.

Apart from this, users **MUST** tamper two of configuration files, one for each service. Below we proceed to nail down which parameters are to be tweaked (some of the attributes present in this file are expected to remain untouched):

1. Parking estimator API

- **Server information:** In this case, the port where the Docker container will be listening at (default 3000) and an attribute called “subscription_host”, which will be the address to which the events from the subscriptions will reference. As a side note, users can choose between development and production environments (the variables are duplicated), so they can easily switch between these two.
- **Keras location:** endpoint where the AI Prediction Module server is.
- **Configuration of the RZs’ Orion Context Brokers.** In an array-like format, we have to manually configure the connection to each Context Broker. Namely, these are the attributes to set:
 - o **rz:** Name of the RZ. This is important because
 - o **orion_host:** Orion CB’s location
 - o **fiware_service:** If the RZ uses this parameter to identify the service/tenant, we must specify it
 - o **fiware_servicepath:** If the RZ relies on this to support hierarchical scopes, we must specify it

¹³ <https://hub.docker.com/r/fiware/orion/>

¹⁴ <https://github.com/docker/compose>

- o **historic_host**: Location of the server to retrieve historical data
 - o **auth**: Boolean that defines whether the
 - o **area**: Rectangle that frames the location of the city. The reason behind this parameter is because an issue with the entity IDs followed by some of the cities, thing that makes impossible to identify an entity from its ID. Hence, from the location, we can know to which RZ it belongs to.
 - **Subscriptions (subscribe_context)**: Enable/disable subscriptions. Note: if the value is “True”, the CB will deliver the notifications to the endpoint defined in the variable “subscription_host” introduced above.
 - **Others**: As hinted before, there are other parameters that have to do with a more generic deployment of the service. Anyway, if we are only targeting to use it as a Parking Estimator, nothing else needs to be tweaked.
2. AI prediction module:
- **time_band**: Time window between which the model will be trained
 - **num_forecasts**: Length of the output array generated by the API (default 4)
 - **train_interval**: Time between consecutive training processes
 - **num_previous_measures**: Length of the array that cater the learning process
 - **minutes_between_measures**: Period of time of the timeseries (time between two consecutive values in the timeseries)
 - **num_features_dataset**: Number of features that shapes the model
 - **column_data**: As the AI prediction modules handles a table-like data input format, we must select, among all the columns (we assume each column gathers a different feature), which one is the target for the predictions
 - **algorithm**: Name of the algorithm used for training the model. Between the two of them we introduced in Section §2.3.4, “*lstm*” is the default option. Apart from it, “*dense*” is the other option we can choose so far¹⁵.
 - **epoch_internal**: Number of iterations the training will be executed for a specific batch size
 - **epoch_external**: Number of times the batch size is increased and the neural network is retrained (default 5)
 - **batch_size**: Number of samples that shape a batch in which the internal epoch is divided
 - **initial_validation_split**: Ratio between testing and training samples (default 5%)
 - **validation_split_multiplier**: Multiplier of the validation split used in each external epoch (default 1.75)

Once we have our configuration files ready, we can proceed to build and run the different Docker images. As an illustrative example, we paste below the code snippet that would yield to the deployment of the tuple Parking Estimator API plus AI prediction module:

```
version: '3'

services:
  ai-prediction-engine:
    container_name: ai-prediction-engine
    image: synchronicityiot/ai-prediction-engine:v0.8
    volumes:
      - ../config:/usr/src/app/config
    environment:
      NODE_ENV: development
    ports:
      - 5000:5000
```

¹⁵ These are the options available at the time of writing this deliverable. Nonetheless, we foresee to append more of them in the short/mid-term.

```
parking-estimator:
  container_name: parking-estimator
  image: synchronicityiot/parking-estimator:v0.8
  depends_on:
    - ai-prediction-engine
  volumes:
    - ../parking-estimator/config:/usr/src/app/config
  ports:
    - 3000:3000
  network_mode: "host"
```

Though mentioned above, it is also worth mentioning that there is a pure Docker way to run (separately) each container, and make it work together. Here, it is important that the Parking estimator API must be deployed after the AI prediction module.

2.3.6 Support

Should the reader or any external developer has any question to ask to the maintainers of this atomic service, the recommended way is to do it via GitLab's issues¹⁶.

2.3.7 License & Terms and Conditions

All the components and micro-services that shape this atomic service are Open Source, under the terms of the GNU General Public Licence as published by the Free Software Foundation (version 3).

For more information and details about the licensing, we encourage the reader to go and check the Synchronicity GitLab repo.

¹⁶ <https://gitlab.com/synchronicity-iot/traffic-flow-estimator/issues>

2.4 Traffic flow estimator atomic service

Being this atomic service a parallel development to that of Parking Estimator, it is deemed necessary to state here the fact that most of the content included in Section §2.3 also applies here. As a matter of fact, the main changes reflected in this deliverable with regards to the Traffic Flow estimator are:

- We have updated the description of the atomic service (Section §2.4.1), where we give a different and more updated outlook to the service itself.
- Some slight modifications were carried out with regards to the API specification (Section §2.4.3.1)
- We also include how the architecture looks like for the Traffic Flow Estimator atomic service (Section §2.4.4)

2.4.1 Description

Together with the Parking Estimator service that we have covered in the previous section, we take advantage of the process of using a fully-fledged AI engine to estimate the value of forthcoming events (in this case, those ones that have to do with traffic intensity). Thanks to the flexible nature we have bestowed to the AI prediction module (at its design phase), we are able to straightforwardly map a new service that, instead of parking data, works with the traffic intensity values measured by these (i.e. TrafficFlowObserved) context elements. Besides, it is worth recalling that both atomic services were conceived to be part of the Multi-modal transportation application theme, which is to be developed and deployed by a handful of Reference Zones (i.e. Santander, Porto, Milano and Carouge).

In terms of the added value for the user, we do offer the capacity to have a rough idea of what will likely happen in the most immediate future. Of course, it goes without saying that all these estimations might not be completely accurate; for instance, an accident may happen at any time and all the prediction made from past values would be little more than worthless.

As a brief summary, this atomic service, from all the knowledge acquired throughout the history of all the deployed traffic flow sensors, we train a model that will be subsequently used to predict the intensity of traffic in the next hour, giving a slotted array of outputs (i.e. 15, 30, 45 and 60 minutes).

Hence, in this section we proceed to span the main highlights we have carried out on the Traffic Flow Estimator atomic service. Nonetheless, as both this one and the parking estimator share most of their elements (they basically share the same architecture and structure), for the sake of avoiding the repetition of text, we will properly refer to the alike Parking Estimator Section wherever corresponds where corresponds.

All the documentation related to this atomic service can be found at:

- Traffic Flow estimator API module - Project's GitLab
- Traffic Flow Estimator API documentation – Apiary [27]
- Traffic Flow Estimator – Docker Hub [28]
- AI prediction module – Docker Hub [21]

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 atomic service.

Table 9 Multi-modal transportation application theme FRs addressed

2.4.3 Interface

2.4.3.1 Traffic Service API

All the modifications performed to update the Parking Estimator API reflected in [1] have been mimicked for this API to, thus leaving these definitive methods:

- **List Traffic Flows:** This method returns an array with the (paginated) list of TrafficFlowObserved context elements, as described in Table 10.
- **Current Traffic Flow Conditions:** instead of yielding the whole list of entities, this method only returns the context information of one of them, specified in the URL. Table 11 gathers how we should call this service.
- **Traffic Flow Conditions Estimation:** Last, but not least, this third method does correspond to the actual output of the AI prediction module. Technically speaking, it returns, aside from the legacy context information of a particular TrafficFlowObserved, a new attribute, called TrafficFlowConditionEstimation, that contains the output of the estimation itself. As a matter of fact, Table 12 reflects the main content of this service, both in terms of input parameters and possible responses).

The draft technical description of the Traffic Service APIs is also available at [27].

API Name		List Traffic Flows			
Description		This service will return the paginated list of the registered observed Traffic Flows.			
End-Point		/trafficflows{?page, items_per_page}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
page	Query	number	No	1	Page number
items_per_page	Query	number	No	-	Number of context elements displayed per page. If nothing is specified, all the elements are displayed
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 10 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 11 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 12 Traffic Flow Conditions Estimation API

2.4.3.2 Information Model

Since the data structures have been subjected to no changes since [1] we invite the interested reader to refer to that document or the published API documentation [27].

2.4.3.3 Interaction diagram (with the external world)

Since the data structures have been subjected to no changes since [1] we invite the interested reader to consult that document.

2.4.4 Service Architecture

Thanks to the way the AI prediction module was designed (i.e. following a micro-services-based approach), the immense majority of the service architecture remains intact (compared to that of Parking estimator, shown in Figure 3 – page 25). Figure 7 illustrates the architecture followed for this atomic service.

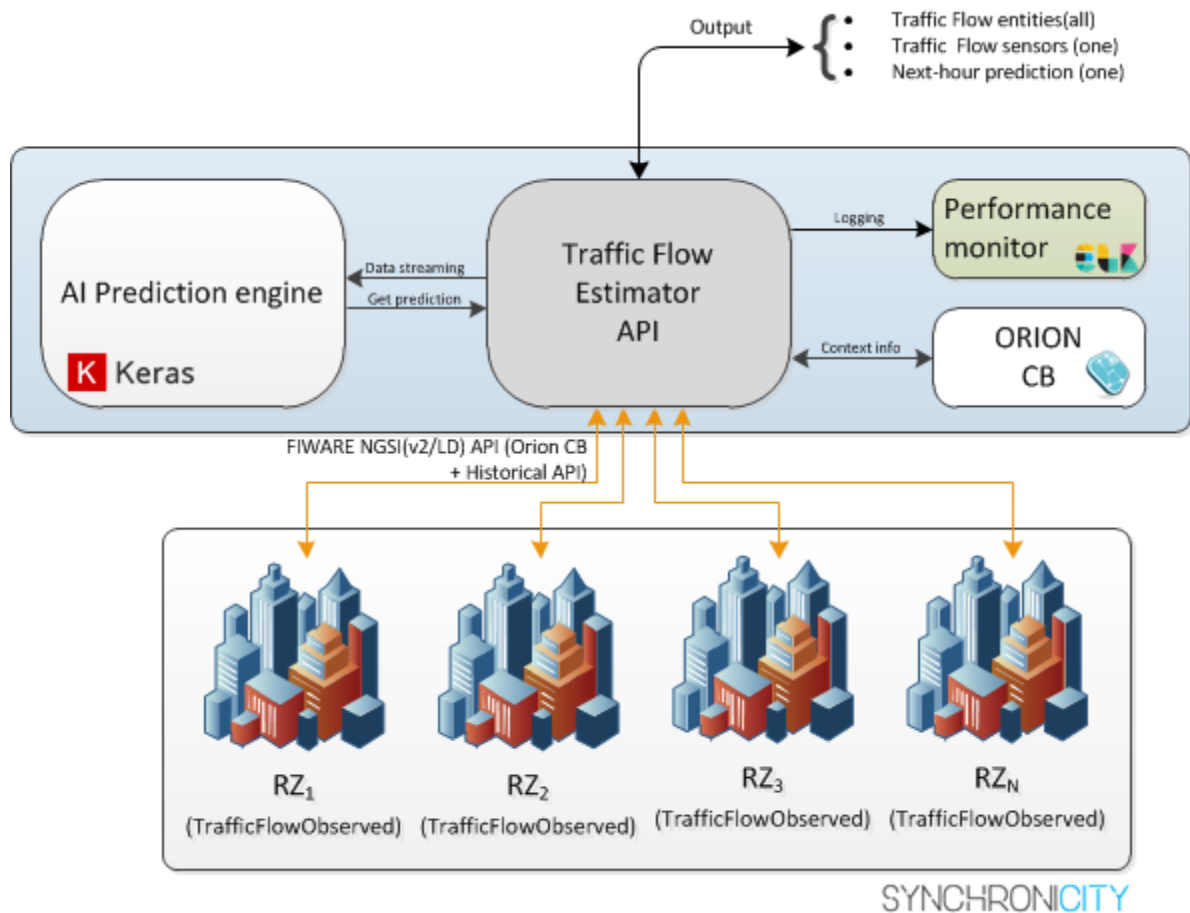


Figure 7 Traffic Flow estimator service architecture and interaction with the SynchroniCity framework

Technically speaking, we only needed to revamp the Estimator API by modifying some attributes at the configuration files defined in Section §2.3.4.

2.4.5 Service Deployment

The deployment process is alike that of described for the Parking Estimator in Section §2.3.5

2.4.6 Support

Should the reader or any external developer has any question to ask to the maintainers of this atomic service, the recommended way is to do it via GitLab's issues¹⁷.

2.4.7 License & Terms and Conditions

All the components and micro-services that shape this atomic service are Open Source, under the terms of the GNU General Public License as published by the Free Software Foundation (version 3).

For more information and details about the licensing, we encourage the reader to go and check the Synchronicity GitLab repo.

2.5 Smartcities Dashboard atomic service

The Smartcities Dashboard atomic service was subject to no changes since [1]. Hence we keep here

¹⁷ <https://gitlab.com/synchronicity-iot/traffic-flow-estimator>

the general description and the value proposition but we invite the reader to such document for more technical information or to the published docker and documentation [29].

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 API¹⁸ 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.

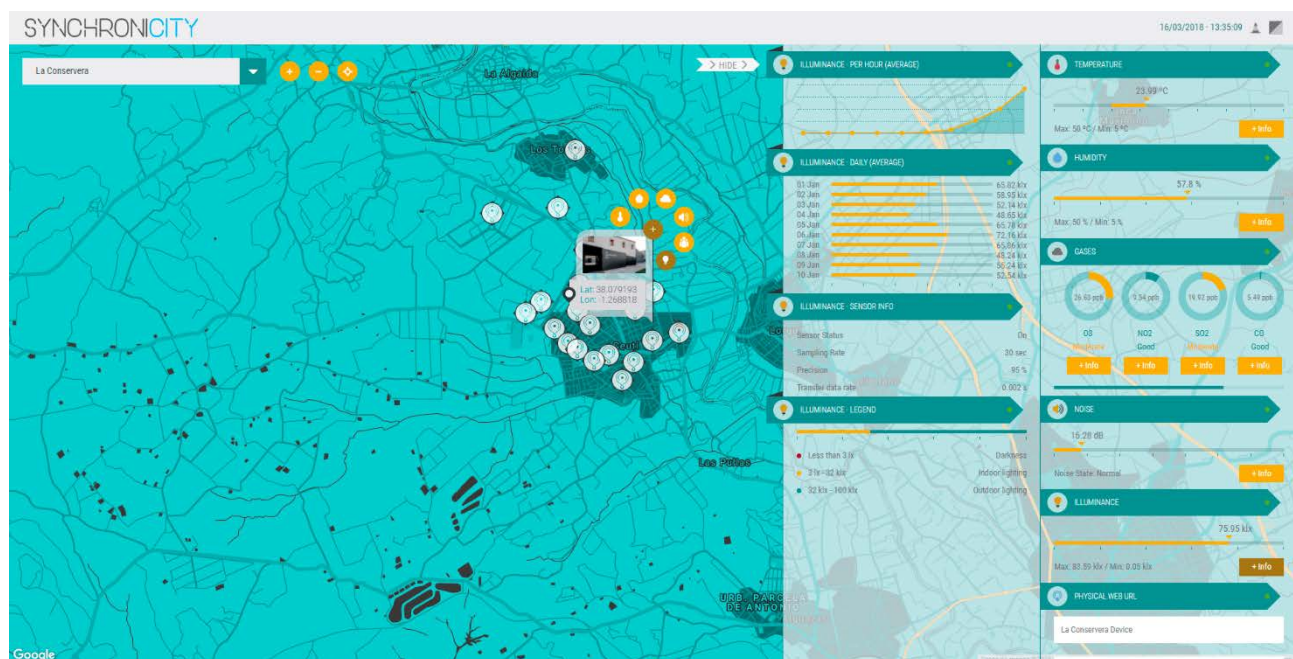


Figure 8 Smartcities Dashboard view

2.5.2 Application theme Functional Requirement addressed

FR code	FR description	Comments
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.

¹⁸ <https://developers.google.com/maps/documentation/>

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 13. FRs of the Human-Centric Traffic Management application theme addressed by the Smartcities Dashboard atomic 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 14 FRs of the Multi-Modal Transportation application theme addressed by the Smartcities Dashboard atomic 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 15 FRs of the Community Policy Suite application theme addressed by the Smartcities Dashboard atomic service

2.5.3 Interface

Since the exposed interface has been subjected to no changes since [1] we invite the interested reader to access that document or the published API [29]

2.5.4 Service Architecture

Since the service architecture has been subjected to no changes since [1] we invite the interested reader to refer to that document.

2.5.5 Service Deployment

Since the service deployment has been subjected to no changes since [1] we invite the interested reader to consult that document. The service is available on the SynchroniCity dockerhub [29]

2.5.6 Support

If there is any reader or external developer who has any question or even they need technical support, the recommended way is to do it via GitHub issue tracker¹⁹.

2.5.7 License & Terms and Conditions

This atomic service is a fully HOPU development and it has a private license. However, it will be free for partner and reference zones involved in SynchroniCity project through the main consortium or Open call.

Any organization which is interested in this service for other project outside SynchroniCity, in order to use it as part of their daily activity in some project or to be deployed in any external city, these organizations should contact with HOPU direction in order to define conditions in terms of fees and services included.

¹⁹ <https://gitlab.com/synchronicity-iot/SmartCitiesDashboardService>

2.6 Metrics Visualizer (Grafana) atomic service

The Metrics Visualizer (Grafana) atomic service was subject to no fundamental change since [1] besides the introduction of a plugin that is a customization to SynchroniCity. Hence we keep here the general description and the value proposition but we invite the reader to consult [1] for more technical information or to consult the published docker and documentation [30].

2.6.1 Description

Metrics Visualizer is a custom deployment of Grafana²⁰ software working on top of FIWARE Stack services.

Through Grafana, a set of databases can be configured as data sources in order to create personalized charts and tables. These databases are filled using QuantumLeap²¹ 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 9.

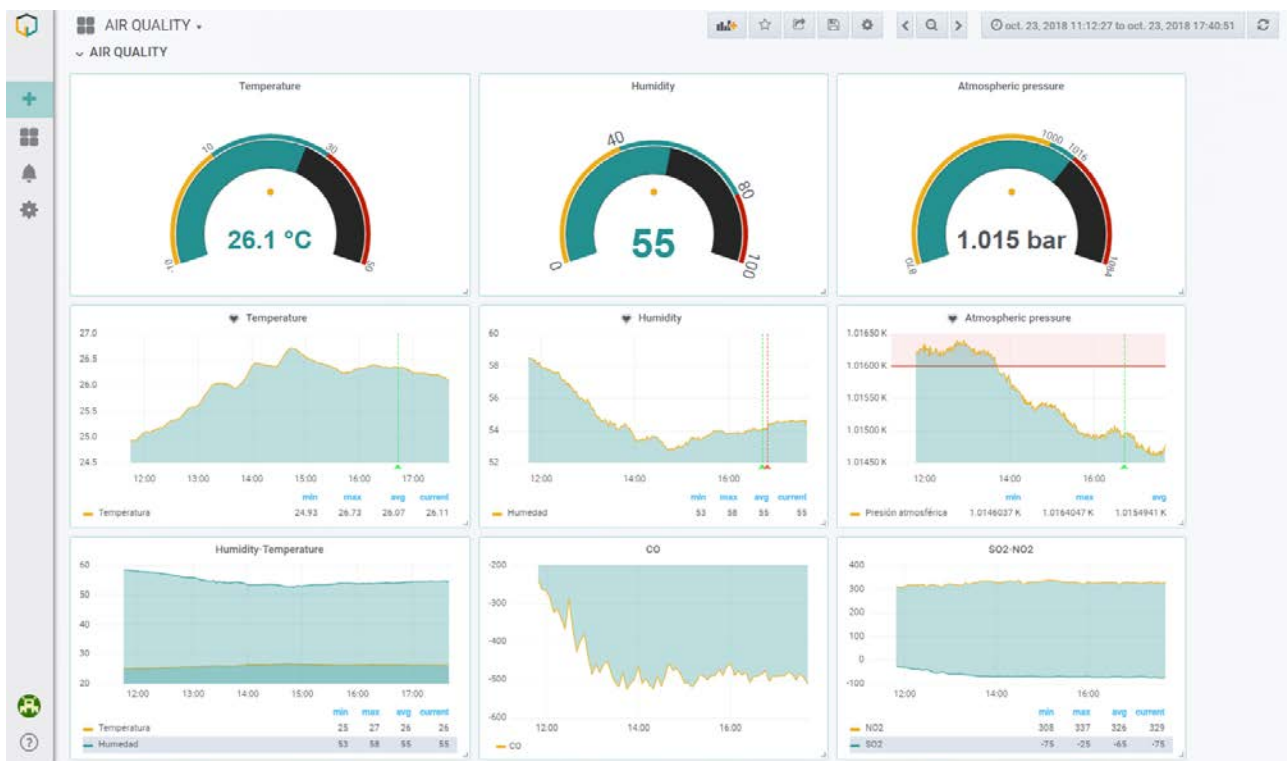


Figure 9 Grafana Dashboard example

In addition one SynchroniCity customized plugins is included:

- Visualization of historical data on a map

²⁰ <https://grafana.com/>

²¹ <https://smartsdk.github.io/ngsi-timeseries-api/>

2.6.2 Application theme Functional Requirement addressed

FR code	FR description	Comments
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 16 FRs of the Human-Centric Traffic Management application theme addressed by the Metric Visualizer (Grafana) atomic 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 17 FRs of the Community Policy Suite application theme addressed by the Metric Visualizer (Grafana) atomic service

2.6.3 Interface

The interface of this atomic service was subject to no updates since [1].

2.6.4 Service Architecture

The service architecture (i.e. service components and operation model) are not changed since [1]. Whereas a new customized Grafana plugin, specific for SynchroniCity, has been developed and made available.

2.6.4.1 Customization for Synchronicity

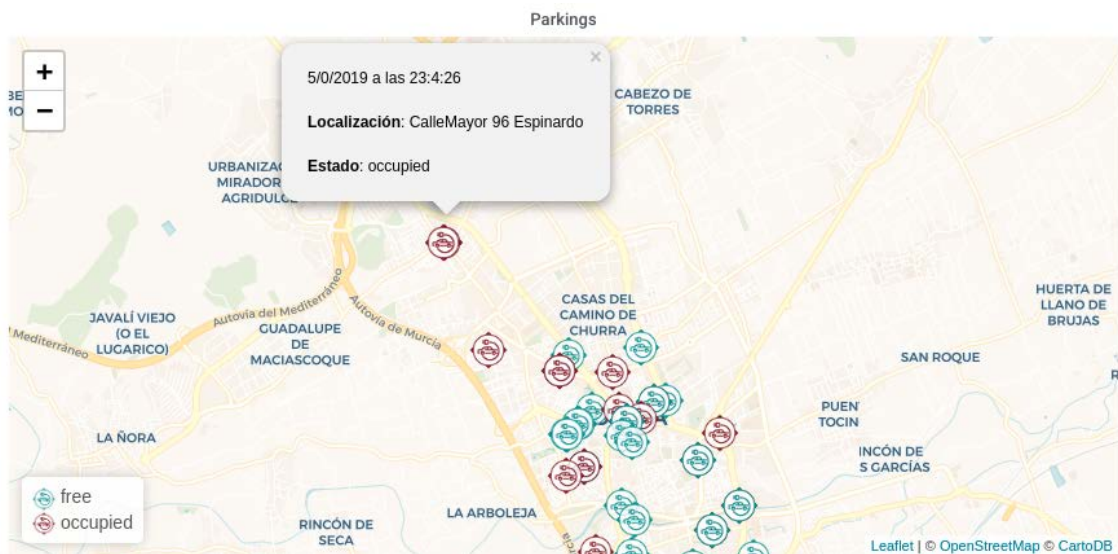


Figure 10 Grafana Worldmap customization for SynchroniCity

Our plugin is based on plugin Worldmap Panel of Grafana²².

In that plugin, it is shown a map on which geographical data is displayed. The data visualized are:

- Longitude and latitude
- Element ID
- Timestamp
- Status field or measure

To obtain the data previously mentioned this plugin needs to send a query to the database which allows users to obtain the attributes desired. To do that, users should access the configuration mode, clicking into the plugin name and selecting “Edit”.

Once the edit mode is opened, users should choose *metric* tag. In this tag is selected the data which had been configured in previous step (1), The format must be “table” (2), then it needs to change to “Toggle Edit Mode” (3) and finally it is necessary indicate the SQL query (4) which must be done to obtain data, as shown in Figure 11.

²² <https://grafana.com/plugins/grafana-worldmap-panel>

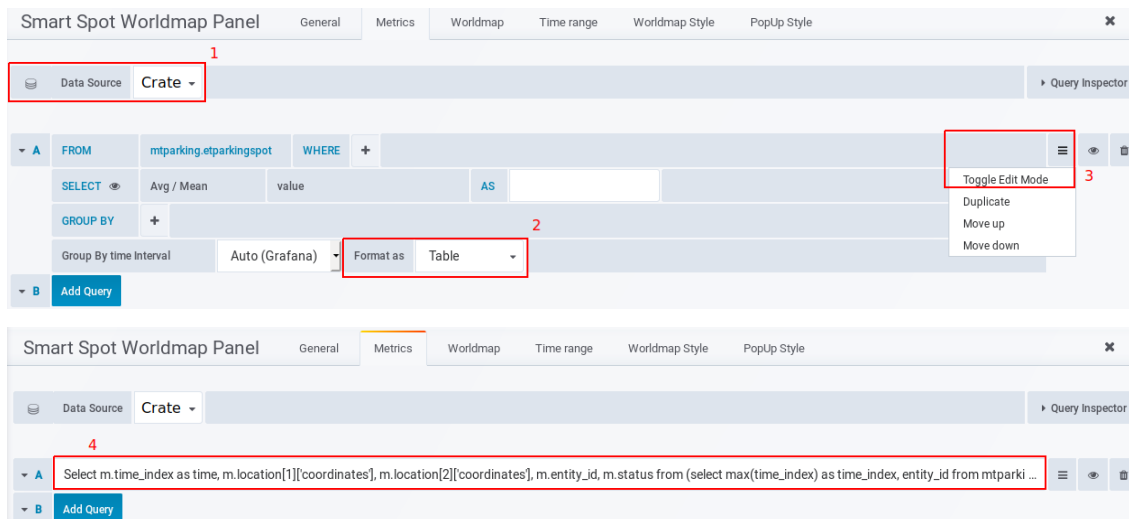


Figure 11 Configuration of the Worldmap plugin

In the following code snippet we show query example to the aforementioned database. This query shows the most updated information of each entity with their attributes longitude, latitude and status (sensor situation), respectively.

```
SELECT
  m.time_index AS time,
  m.location[1]['coordinates'],
  m.location[2]['coordinates'],
  m.entity_id,
  m.status
FROM (
  SELECT
    max(time_index) AS time_index,
    entity_id
  FROM mtparking.etparkingspot
  GROUP BY entity_id ) t
JOIN mtparking.etparkingspot m
ON m.time_index = t.time_index
AND m.entity_id = t.entity_id
limit 100;
```

All of this is similar to Worldmap Panel. However, the developed plugin allows us to modify the map layout. In Figure 12 we show different options for the map appearance.

Figure 12 Metrics Visualizer (Grafana) – Worldmap plugin: Representation Options

In this section is mentioned the type of data which must be represented (it could be numeric or string).

In the case of numeric data, data will be represented in a scale of colours; in case of string the output can be discretized with cases, and users can choose to represent them with either circles or icons (via URL).

OnClick Option

Figure 13 Worldmap "onclick" option

If there is a general Grafana variable configured, user can indicate the variable name (e.g. *parking_place*) in this field, and each time the map element is clicked, the variable will have assigned the element ID of the element which is represented.

Section Worldmap tile

Figure 14 Worldmap tile options

In this section is configured the topic map and there are two possibilities:

1. Select: Choice a pre-configured theme.
2. InsertParams: users might indicate through an URL a different theme which it not available in Grafana (several other examples can be found in [31]).

Selection PopUp Style

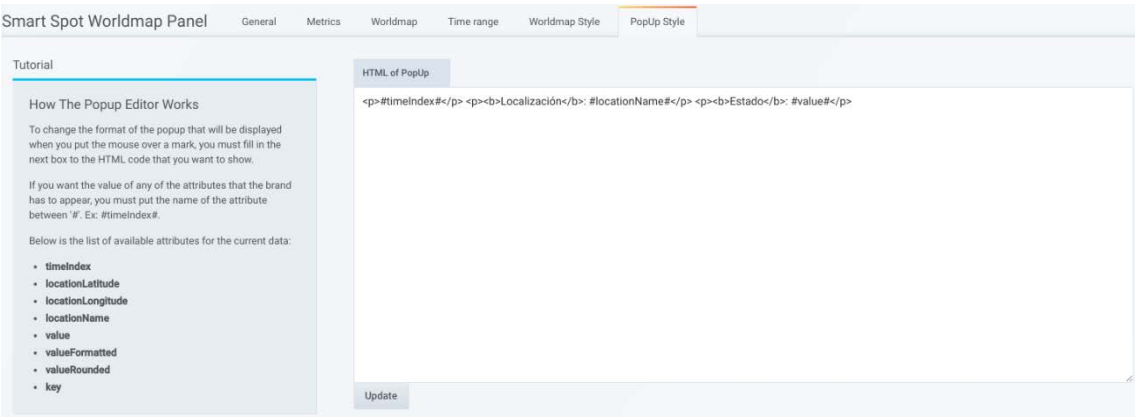


Table 18 Worldmap plugin PopUps configuration

The PopUp might be drawn clicking over a mark into the map. In this section, there is an editor in which it can be configured the HTML format with which users want to show the PopUp information. At the left of the panel, there is a list of possible attributes that can be used in the HTML code. To show the values, it is necessary to write the attribute between hash (“#”) symbols. For instance, “#timeIndex#”, as shown in the following code snippet:

```
<p>#timeIndex#</p> <p><b>Localización</b>: #locationName#</p> <p><b>Estado</b>: #value#</p>
```

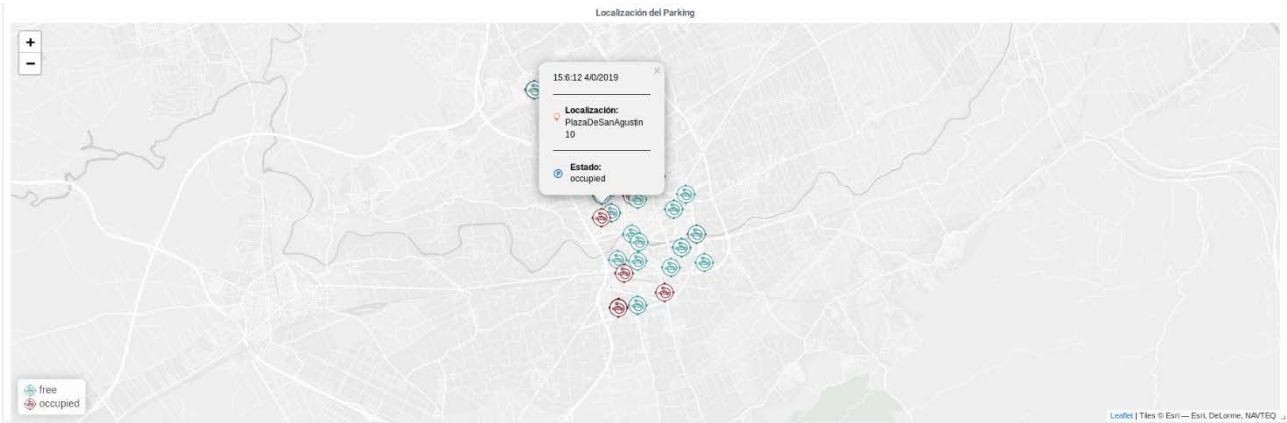


Figure 15 Customized plugin showing parking items

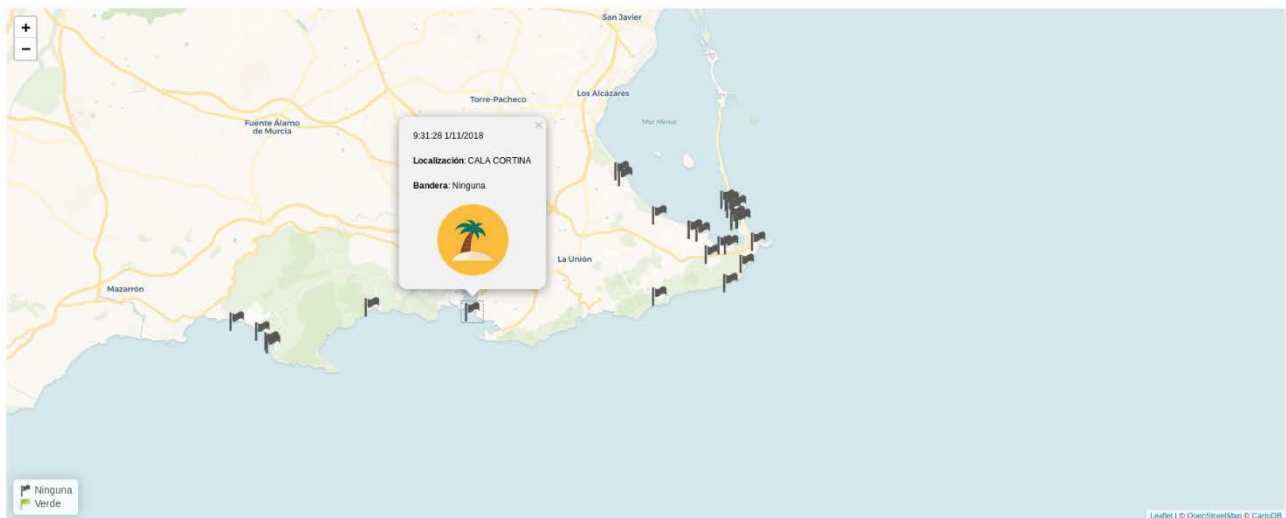


Figure 16 Customized plugin showing beaches items

2.6.5 Service deployment

Since the service deployment has been subjected to no changes since [1] we invite the interested readers to consult that document. The service is available on the SynchroniCity dockerhub [30].

2.6.6 Support

If there is any reader or external developer who has any question or even, they need technical support, the recommended way is to do it via GitLab issue tracker²³.

2.6.7 License & Terms and Conditions

Grafana is a publicly distributed visualization engine for time series data which has been adapted to Synchronicity framework. Although, Grafana is not a tool developed by Synchronicity community, there are several plugins developed by HOP Ubiquitous, integrated and ready to be used into the Synchronicity framework. For that reason, license & terms and conditions are those offered by Grafana²⁴ for those developments that have been done by Grafana community.

For further information and details about the use, we encourage the reader to go and check the SynchroniCity GitLab repository²⁵

²³ <https://gitlab.com/synchronicity-iot/GrafanaAtomicService>

²⁴ <https://github.com/grafana/grafana/blob/master/LICENSE.md>

²⁵ <https://gitlab.com/synchronicity-iot/GrafanaAtomicService>

2.7 GTFS Fetcher atomic service

This GTFS Fetcher is a brand new atomic service. This component is necessary in order to integrate the Routing service with the SynchroniCity framework.

2.7.1 Description

GTFS static data is widely available in text files packed on a compressed single (e.g. zip) file. Tests have been made to load such data into Orion, and it was concluded that such approach was not only inefficient, but also difficult to make it adopted by clients (like openTripPlanner).

Given that and continuing to pursuit the goal of having Orion as a source of information, it was decided that a new data model should be created to store information about the availability of GTFS data on another location. That is, to have instances on Orion Context Broker announcing that GTFS files exists in a specified location, for transportation service providers.

This service provides a mechanism to fetch GTFS files (static timetables and related info), based on availability data of such files announced within the SynchroniCity framework (i.e. FIWARE Orion). The mentioned availability data is retrieved from the Context Broker via NGSI. The service is then handling the fetched GTFS in various ways, e.g. feeding to an application or storing it somewhere. How the GTFS file needs to be handled is specified by an extensible system of plugins.

Two plugins for OpenTripPlanner are included in the roadmap, one for OpenTripPlanner is running on the same container, and another one for OpenTripPlanner running on a different container (with shared volumes).

2.7.2 Application theme Functional Requirement addressed

This service does not address any Functional Requirement directly. Instead it provides a way to refresh data provided by routing service (like OpenTripPlanner).

So, indirectly, addresses SynchroniCity FRs such the one shown in Table 19.

FR code	FR description	Comments
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).	This service allows the Routing Atomic Service to recognize PoI and other geographical items.
FR-12	The user should be able to view on a map the points of interest located in an area near to the route.	GTFS Fetcher allows to provision PoI encapsulated in GTFS to GTFS compliant services (such as Routing Services)

Table 19 Multi-modal transportation application theme FRs addressed

2.7.3 API

This service does not implement an API. It works continuously and acts, not by request, but on the presence/update of specific entities in Orion.

2.7.4 Service Architecture

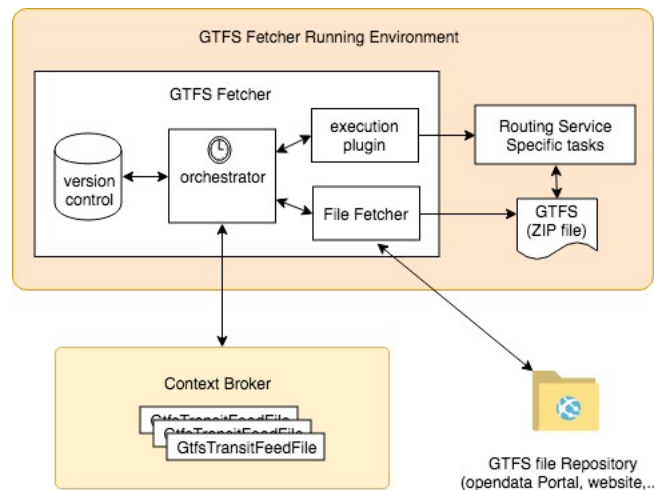


Figure 17 – GTFS Fetcher Service architecture and basic running environment

As stated in the diagram above, this service's core is an orchestrator which calls the required procedures.

Orchestrator – This component is responsible for fetching *GTFSTransitFeedFile* type entities from Orion Context Broker, filtered according to the service configuration. For each entity retrieved, it starts by comparing entity's 'version' attribute against an internal database, holding GTFS Versions previously downloaded. If entity's version is accountable, then it performs the operations implemented in the following components (by this order):

File Fetcher – fetches/downloads GTFS zip file located in the URL specified in the 'feedFileURL' attribute, to a location specified in the configuration;

Plugin Executor – Executes a plugin. Typically it is an executable script that performs operations specific for the client application (for instance, OpenTripPlanner).

As an example, for Open Trip Planner, the plugin should trigger graph build and, having success, should call OTP API to reload graph (in case autoReload is not configured).

2.7.5 Implementation Roadmap

This is a simple service and so its implementation started as soon as the initial requirements were assessed.

Nevertheless, an implementation roadmap was drafted, with special attention on testing different scenarios ranging integration with different versions of OpenTripPlanner and methods (use OTP's graph autoreload, use OTP API), to different deployment scenarios (same docker container as OTP vs own container containers).

- Initial analysis, requirements and architecture definition
 - o Analysis - DONE
 - o Code snippet implementation to test concepts – DONE
 - o Proposal Definition for the "GtfsTransitFeedFile"
- Base service (without plugins)
 - o Implementation - DONE
 - o Testing - DONE

- Plugin for OpenTripPlanner on single container
 - o Implementation - DONE
 - o Testing - DONE
- Plugin for OpenTripPlanner on a separate container
 - o Implementation - ONGOING
 - o Testing
- Cleanup, Final Packing and Documentation

As already mentioned, the “GTFSTransitFeedFile” datamodel exists as a proposal, and is expected to be approved soon.

Plans are made to develop additional features such as a management API (to allow changing configuration on-the-fly, without having to change the running container, for instance)

2.7.6 Support

Support to this component will be handled by its developers in a best effort basis. That said, any issues or contributions should be addressed to project GIT repository²⁶.

2.7.7 License & Terms and Conditions

The code will be released under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

²⁶ To be defined/created in gitlab synchronicity's group repository

2.8 NGSI Urban Mobility to GTFS atomic service

This NGSI Urban Mobility to GTFS is a brand new atomic service. This component is necessary in order to integrate the Routing service with the SynchroniCity framework.

2.8.1 Description

The representation given by Urban Mobility data models, defined within FIWARE Data Models [32] and adopted by SynchroniCity [33], is closely related to the specific nature of the NGSI Context information. Urban Mobility data models have been designed to provide a mapping to NGSI representation of the GTFS feeds, enabling the usage and storage of that data as context information in a Context Broker.

The General Transit Feed Specification (GTFS static) [34] defines a common format for public transportation schedules and related geographic information. A GTFS feed can be used by tool developers, in order to exploit the public transit information (routes, trips, stop, etc.) issued by several transport agencies, in an interoperable way. Applications consuming these GTFS feeds, will be able to realize several functionalities, such as trip planning and time table publishing.

Each Urban Mobility NGSI entity (e.g. `gtfs:Agency`, `gtfs:Route`) has been defined by mapping the corresponding entities compliant to GTFS [8] and LinkedGTFS [35] specifications: Agency, Routes, Trips, Stops, etc.

The Urban Mobility to GTFS service is in charge of converting NGSI entities, compliant to Urban Mobility Data Models, to the native GTFS feed format, performing the inverse mapping done when defining the NGSI counterpart. Therefore, it will take in input entities to be mapped by querying directly a NGSI Context Broker (e.g. trips belonging to a specific route, issued by a specific transport agency), and then it will return the translation as GTFS feed of the related entities in a compressed archive.

The service can be used either as on-demand translation tool, using the dedicated API (described in next sections), or as a daemon service, which at fixed time interval will automatically retrieve NGSI entities and create (if entities have been updated) a new GTFS feed archive. In the first case, the feed will be directly returned as a binary file; in the second one, the tool will write the resulting archive file to a specific internal or remote directory, in order to be accessed asynchronously by external systems.

In addition, the service will keep updated, in the Context Broker, a specific NGSI entity compliant to `GTFSTransitFeedFile` (introduced and described in Section §2.7). This entity provides metadata about the resulting GTFS Feed, such as the location (URL) of where the ZIP file containing the feed is stored and available. This process realizes the connection between this service and the GTFS Fetcher atomic service, described in Section §2.7. That service, indeed, requires the `gtfsTransitFeedFile` entity to get information about the availability of GTFS data on another location, such as the ones managed by the Urban Mobility to GTFS service.

2.8.2 Application theme Functional Requirement addressed

This service does not directly address any Functional Requirement, but it supports other tools and services to address some specific requirements.

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

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).	This service allows the Routing Atomic Service to recognize PoI and other geographical items, such as stations, bus stops, etc, provided by a NGSI compliant data source (such as Orion Context Broker).
FR-12	The user should be able to view on a map the points of interest located in an area near to the route.	This service allows to provision PoI (e.g.: station, bus stops, etc.) described in NGSI as GTFS files to GTFS compliant services (such as Routing Services)
FR-13	The application may consider real-time information (deleted rides, delays, etc.) to better plan and update the journey	This service offers to application not compatible to NGSI a convenient way to get data about transportation from NGSI compliant sources (such as Orion Context Broker).

Table 20 Multi-modal transportation application theme FRs addressed

2.8.3 Interface

2.8.3.1 Urban Mobility to GTFS API

This section provides a first description of the Urban Mobility to GTFS API, focused on its main functionalities. The final implementation of the APIs will be provided in the future in the published documentation²⁷.

The Urban Mobility to GTFS API is composed by two sets of RESTful services:

- Urban Mobility Query API
- GTFS Translation API

The Urban Mobility Query API allows to query for NGSI Urban Mobility entities, available in a configured Context Broker, according to a hierarchical level of granularity, for instance: all the routes and trips issued by a specific transport agency, or all the trips belonging to a specific route. The GTFS Translation API will do the actual translation work. The results returned by Query API (in particular their identifiers), will serve as input to the Translation API, which will translate the corresponding entities to GTFS feeds, returning the resulting archive.

The first version of Urban Mobility Query APIs consists of the following RESTful services:

- `getAgencies`: this method will return the list of available `gtfs:Agency` entities.
- `getRoutes`: this method will return the list of available `gtfs:Route` entities, operated and provided by one or more `gtfs:Agency`.
- `getTrips`: this method will return the list of available `gtfs:Trip` entities, belonging to a specific `gtfs:Route`.

It is important to underline that the previous methods provide a direct way to retrieve and select which entities are to be mapped from those present in the Context Broker; indeed, it would be possible to retrieve the same information using directly the NGSI APIs of the Context Broker.

The first version of GTFS Translation APIs consists of the following RESTful service:

- `getGTFSByAgency`: this method will return the translation as GTFS feed containing all the routes and relative entities, such as trips, stop times and so on, issued by one or more `gtfs:Agency`.

In the following, introduced services will be detailed in summary tables. Furthermore, a short description of data structures adopted in the APIs is also reported.

²⁷ <https://ngsiurbanmobilitytogtfsapi.docs.apiary.io/>

API Name		Urban Mobility Query API - Get Agencies			
Description		This service will return the (paginated) list of the available Urban Mobility Agencies (gtfs:Agency).			
End-Point		/urbanmobility/agencies{?rows,offset}			
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 number of returned results.
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					
HTTP Status Code		Content-type	Entity		Description
200		application/json	Array of gtfs:Agency		The list of the Agencies.
400		application/json	ErrorResponse		Bad Request
500		application/json	ErrorResponse		Internal Server error

Table 21 - Get Agencies API

API Name		Urban Mobility Query API - Get Routes			
Description		This service will return the (paginated) list of the available Urban Mobility Routes (gtfs:Route), issued by one or more gtfs:Agency.			
End-Point		/urbanmobility/routes{?rows,offset}			
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 number of returned results.
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					
HTTP Code	Status	Content-type	Entity		Description
200		application/json	Array of gtfs:Route		The list of the Routes.
400		application/json	ErrorResponse		Bad Request
500		application/json	ErrorResponse		Internal Server error

Table 22 - Get Routes API

API Name		Urban Mobility Query API - Get Trips			
Description		This service will return the (paginated) list of the available Urban Mobility Trips (gtfs:Trips), belonging to a specific gtfs:Route.			
End-Point		/urbanmobility/trips{?rows,offset}			
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 number of returned results.
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					
HTTP Code	Status	Content-type	Entity		Description
200		application/json	Array of gtfs:Trip		The list of the Trips.
400		application/json	ErrorResponse		Bad Request
500		application/json	ErrorResponse		Internal Server error

Table 23 - Get Trips API

API Name		GTFS Translation API - Get GTFS By Agency			
Description		This service will return the GTFS feed archive containing the translation of all the routes (gtfs:Route) and related entities, issued by one or more gtfs:Agency, such as: gtfs:Trip, gtfs:StopTime, gtfs:Stop, gtfs:Service, etc. If the specific Agency is not specified in input, the service will return the GTFS archive containing all the translated entities issued by all the Agencies.			
End-Point		/gtfs/agency/{agencyID}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
agencyID	Path	string	No	None	This is the identifier of a gtfs:Agency, retrieved with the dedicated Urban Mobility Query API. (Table 21). If not provided, the service will consider all the Agencies.
Responses					
HTTP Code	Status	Content-type		Entity	Description
200		application/octet		GTFSFeed	The Zip file containing the GTFS Feed.
400		application/json		ErrorResponse	Bad Request
500		application/json		ErrorResponse	Internal Server error

Table 24 - Get GTFS By Agency

2.8.3.2 Data Structures

This section provides the description of the entities used in the Urban Mobility to GTFS APIs.

The entities reported in this section are:

- gtfs:Agency, gtfs:Route, gtfs:Trip: these NGSI entities, discovered through the Urban Mobility Query APIs, are the entities from which a GTFS feed is created.
- GTFSFeed: this entity models the GTFS Feed archive (ZIP file) containing the files resulting from the translation of corresponding Urban Mobility NGSI entities, done through the GTFS Translation API.
- ErrorResponse: this entity models error messages that may occur during the execution of a request.

For further details about the mapping between NGSI Urban Mobility entities and GTFS Feed files,

see Section §2.8.3.3.

Entity Name	gtfs:Agency	
Description	This entity, mapped from the corresponding GTFS member, agency.txt [36], represents the transportation agency that provides data relative to one or more transit schedule. For more details, please refer to Agency data model specification [37].	
Attributes		
Name	Type	Description
id	String	Entity identifier
type	String	It must be equal to gtfs:Agency
dataProvider	URL	URL pointing to the provider of this information
dateCreated	DateTime	Entity's creation timestamp
dateModified	DateTime	Entity's last modify timestamp
source	URL	Original source of the entity data as a URL. It must point to the URL of the original GTFS feed used to generate it
name	String	Full name of the transit agency
page	URL	URL of the transit agency
timezone	String	Timezone where the transit agency is located
phone	String	Single voice telephone number for the specified agency
language	String	IETF BCP 47 language code specifying the primary language used by the transit agency
address	PostalAddress [38]	Agency's civic address

Table 25 gtfs:Agency entity

Entity Name	gtfs:Route	
Description	This entity, mapped from the corresponding GTFS member, routes.txt [39], represents the transit route, intended as a group of trips that are displayed to riders as a single service. For more details, please refer to Route data model specification [40].	

Attributes		
Name	Type	Description
id	String	Entity identifier
type	String	It must be equal to gtfs:Route
dataProvider	URL	URL pointing to the provider of this information
dateCreated	DateTime	Entity's creation timestamp
dateModified	DateTime	Entity's last modify timestamp
source	URL	Original source of the entity data as a URL.
shortName	String	Short name of the route
name	String	Full name of the route
description	String	Useful description of the route
routeType	String	The type of transportation used on the route. Allowed values as prescribed by GTFS specification for route_type field [39]
page	URL	URL of a web page about the route
routeColor	String	In systems that have colors assigned to routes, the route_color field defines a color that corresponds to a route. Allowed values as prescribed by GTFS specification for route_color field [39]
routeTextColor	String	Legible color to use for text drawn against a background of routeColor. Allowed values as prescribed by GTFS specification for route_text_color field [39]
operatedBy	Number	Identifier of transport Agency (gtfs:Agency entity) that operates this route

Table 26 gtfs:Route entity

Entity Name	gtfs:Trip	
Description	This entity, mapped from the corresponding GTFS member, trips.txt [41], represents the trip composing a specific route. A trip is a sequence of two or more stops that occurs at specific time. For more details, please refer to Trip data model specification [42].	
Attributes		
Name	Type	Description

id	String	Entity identifier
type	String	It must be equal to gtfs:Trip
dataProvider	URL	URL pointing to the provider of this information
dateCreated	DateTime	Entity's creation timestamp
dateModified	DateTime	Entity's last modify timestamp
source	URL	Original source of the entity data as a URL.
shortName	String	The text that appears in schedules and sign boards to identify the trip to passengers
direction	String	A binary value that indicates the direction of travel for a trip. See GTFS specification for direction_id field [41]
block	String	The block to which the trip belongs. A block consists of a single trip or many sequential trips made using the same vehicle. See GTFS specification for block_id field [41]
hasService	String	ID that uniquely identifies a set of dates when service is available for one or more routes. It must point to a gtfs:Service entity [43]
location	GeoJSON	The geographical shape associated to the trip encoded as GeoJSON LineString or MultiLineString. The GTFS counterpart relates to shape_id, whose NGSI mapping is still under development. Thus, currently the GeoJson is reported directly in this field
hasRoute	String	The identifier of a gtfs:Route entity to which this trip refers to.
wheelChairAccessible	String	Accessibility information for the trip. Indicates if riders in wheelchairs can be accommodated on this trip. Allowed values as prescribed by GTFS specification for wheelchair_accessible field [41]
bikesAllowed	String	Indicates if bicycles are allowed for this trip. Allowed values as prescribed by GTFS specification for wheelchair_accessible field [41]

Table 27 gtfs:Trip entity

Entity Name	ErrorResponse
Description	This entity is used to provide information about errors occurred during the execution of a request through the Urban Mobility to GTFS 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 28 – ErrorResponse entity

Entity Name	GTFSFeed	
Description	This entity represents public transportation schedules and associated geographic information in the form of GTFS Feed archive (ZIP file), containing the files resulting from the translation of corresponding Urban Mobility NGSI entities. Each attribute defined below refers to a file contained in the archive, namely a GTFS feed member, each of them will be detailed in following tables. For more details, please refer to GTFS Specification [8].	
Attributes		
Name	Type	Description
agency.txt	CSV File	One or more transit agencies that provide the data in this feed. See GTFS Specification [36].
routes.txt	CSV File	Transit routes. A route is a group of trips that are displayed to riders as a single service. See GTFS Specification [39].
trips.txt	CSV File	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time. See GTFS Specification [41].
stops.txt	CSV File	Individual locations where vehicles pick up or drop off passengers.. See GTFS Specification [44].
stop_times.txt	CSV File	Times that a vehicle arrives at and departs from individual stops for each trip. See GTFS Specification [45].
calendar.txt	CSV File	Dates for service IDs using a weekly schedule. See GTFS Specification [46].
calendar_dates.txt	CSV File	Exceptions for the service IDs defined in

		the calendatr.txt file. See GTFS Specification [47].
fare_rules.txt	CSV File	Rules for applying fare information for a transit organization's routes. It specifies how fares in fare_attributes.txt apply to an itinerary. See GTFS Specification [48].
fare_attributes.txt	CSV File	See GTFS Specification [48].
shapes.txt	CSV File	The physical path that a vehicle takes within a route. See GTFS Specification [49].
frequencies.txt	CSV File	Headway (time between trips) for routes with variable frequency of service. See GTFS Specification [50].
transfers.txt	CSV File	Rules for making connections at transfer points between routes. See GTFS Specification [51].

Table 29 GTFS Feed Entity and contained files

All required files in the GTFS Feed are translated from the dual NGSI entity and included in the archive, following the hierarchy defined by GTFS specification, and reported also in following section.

2.8.3.3 Information Model and NGSI-GTFS mapping

This section describes the overall model of GTFS Feed used in the Translation API and how its entities map to the Urban Mobility data models [52].

Figure 18 depicts the logical model of GTFS Feed members (namely tables).

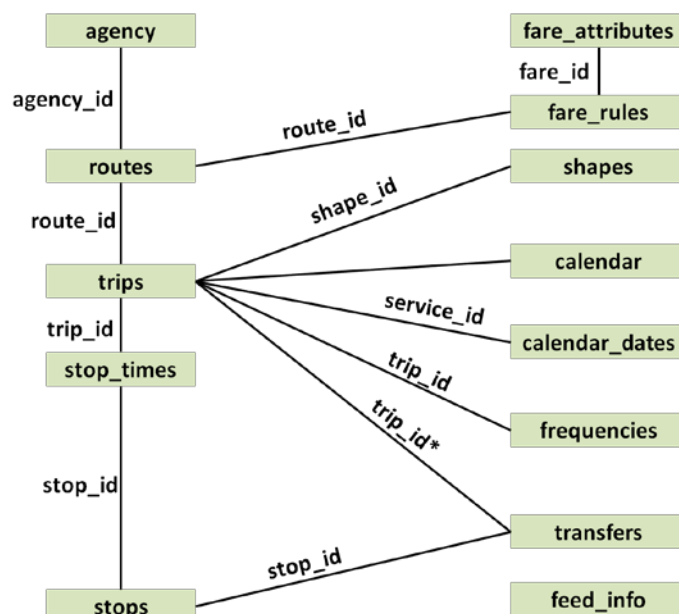


Figure 18 - GTFS data model diagram

As previously introduced, each entity of this model has been mapped into the NGSI Urban Mobility Data Model.

Table 30 summarizes how GTFS tables match with Urban Mobility entities. Mappings reported as N/A denote that corresponding NGSI entities are still under development.

GTFS Feed Tables	NGSI Entity Type
agency.txt	gtfs:Agency
stops.txt	gtfs:Stop, gtfs:Station, gtfs:AccessPoint
routes.txt	gtfs:Route
trips.txt	gtfs:Trip
stop_times.txt	gtfs:StopTime
calendar.txt	gtfs:CalendarRule
calendar_dates.txt	gtfs:CalendarDateRule
fare_attributes.txt	N/A
fare_rules.txt	N/A
shapes.txt	N/A (Coordinates embedded in gtfs:Trip location field)
frequencies.txt	gtfs:Frequency
transfers.txt	gtfs:TransferRule
feedinfo.txt	N/A

Table 30 - GTFS members - NGSI entities mapping

The GTFS Translation API, following the hierarchy depicted in Figure 18, retrieves all the linked NGSI counterparts, in order to translate them to GTFS members and include in the final GTFS Feed.

In the following, we describe the mapping of relevant fields of the main entities involved in the service, in particular the ones used in Translation API.

Agency	
GTFS Field	NGSI attribute
agency_name	name
agency_url	page
agency_timezone	timezone
agency_phone	phone
agency_lang	language

Table 31 - Agency mapping

Route	
GTFS Field	NGSI attribute
route_short_name	shortName
route_long_name	name
route_type	routeType
route_desc	description
route_url	page
route_color	routeColor
route_text_color	routeTextColor

Table 32 - Route mapping

Trip	
GTFS Field	NGSI attribute
trip_headsign	headSign
trip_short_name	shortName
direction_id	direction

shape_id	location
block_id	block
wheelchair_accessible	wheelchairAccessible
bikes_allowed	bikesAllowed

Table 33 - Trip mapping

Furthermore, in the following are described the specific fields that denote relationships between previously described entities.

Route		
GTFS Field	NGSI attribute	Comment
agent_id	operatedBy	Points to a gtfs:Agency entity

Table 34 - Route relationship fields

Trip		
GTFS Field	NGSI attribute	Comment
route_id	hasRoute	Points to a gtfs:Route entity
service_id	hasService	Points to a gtfs:Service entity

Table 35 - Trip relationship fields

According to Urban Mobility specification [43], the NGSI gtfs:Service entity represents a transportation service which is available for one or more routes (intended as gtfs:Trip) at certain dates. It is referenced by the hasService field of gtfs:Trip and has been defined to group a set of gtfs:CalendarRule and gtfs:CalendarDateRule belonging to a specific trip.

2.8.3.4 Interaction Diagram

This section describes the interactions between a generic Client Application and the Urban Mobility to GTFS service, considering current draft version of the service API, described in §2.8.3.1.

Furthermore, the final diagram describes internal interactions in the specific temporized job performed by the Service Controller component. The components acting in these interaction diagram are described in the architecture section (see Section §2.8.4)

Urban Mobility Query API

These APIs provide the list of Urban Mobility NGSI entities available in the platform. The pagination of returned lists is determined by the input parameters: rows, offset. The following diagrams depict the interactions for each case: gtfs:Agency, gtfs:Route and gtfs:Trip.

GetAgencies API

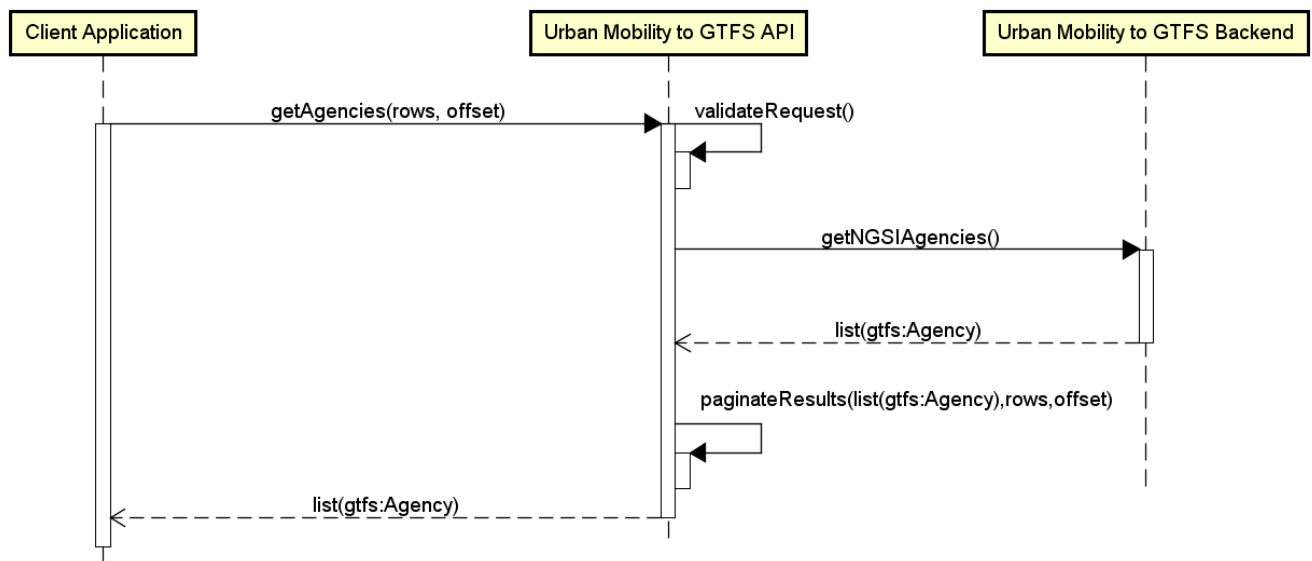


Figure 19 - *getAgencies Sequence Diagram*

The Client Application requests for the available Urban Mobility “gtfs:Agency” entities to the Urban Mobility Query API. The component validates the input request and forward it to the Urban Mobility to GTFS Backend which will retrieve the available “gtfs:Agency” NGSI entities. The resulting list will be paginated according to the input parameters (rows, offset) and returned to the Client Application.

GetRoutes API

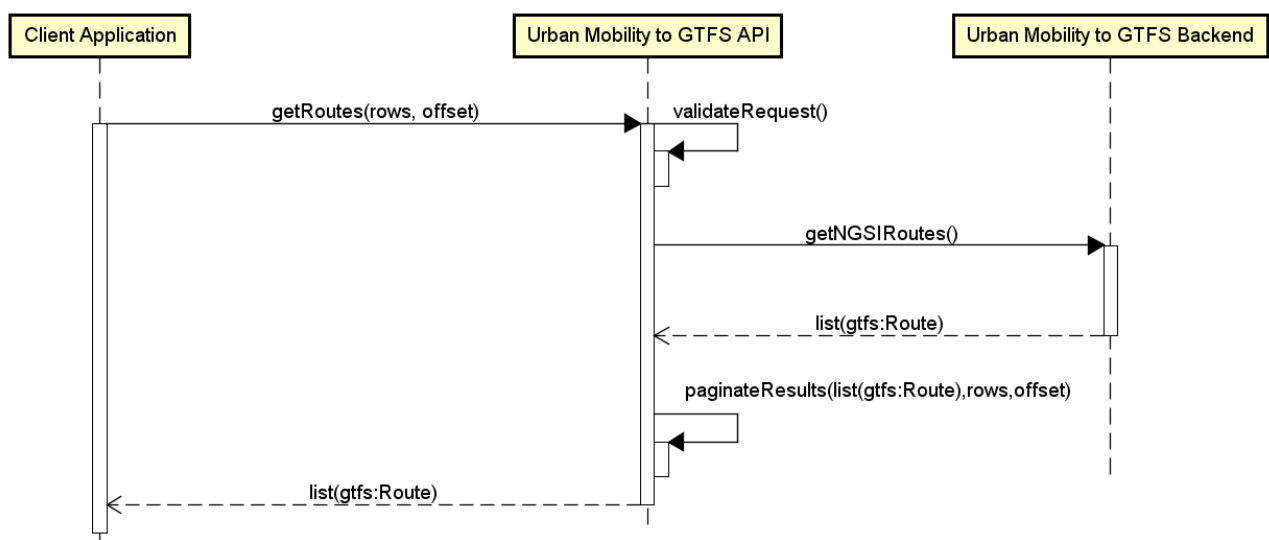


Figure 20 - *getRoutes Sequence Diagram*

The Client Application requests for the available Urban Mobility “gtfs:Route” entities to the Urban Mobility Query API. The component validates the input request and forward it to the Urban Mobility to GTFS Backend which will retrieve the available “gtfs:Route” NGSi entities. The resulting list will be paginated according to the input parameters (rows, offset) and returned to the Client Application.

GetTrips API

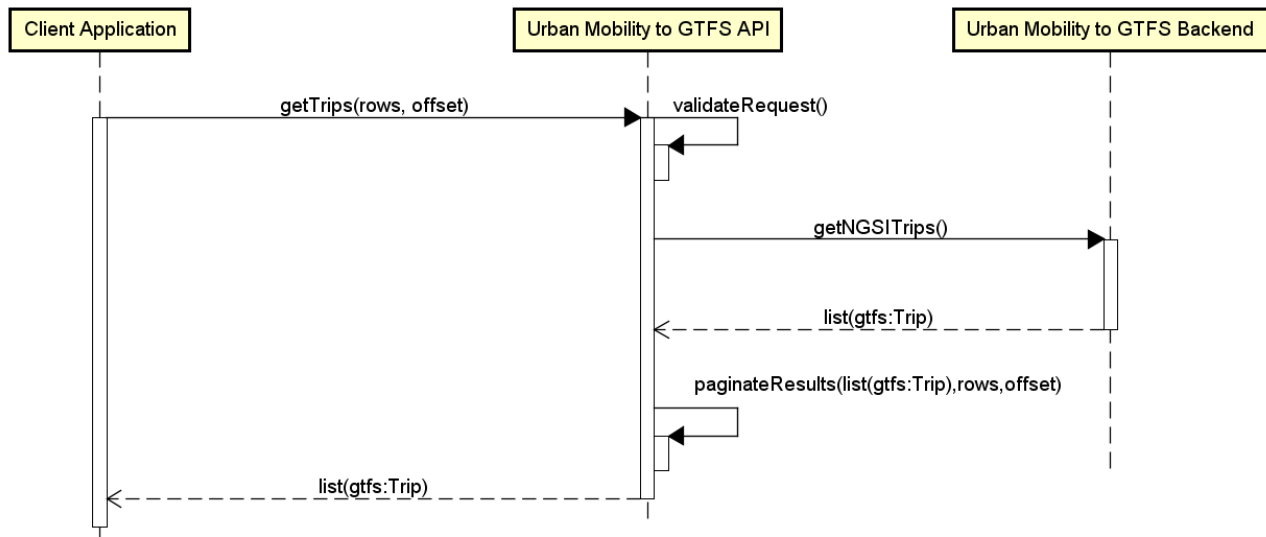


Figure 21 - *getTrips* Sequence Diagram

The Client Application requests for the available Urban Mobility “gtfs:Trip” entities to the Urban Mobility Query API. The component validates the input request and forward it to the Urban Mobility to GTFS Backend which will retrieve the available “gtfs:Trip” NGSi entities. The resulting list will be paginated according to the input parameters (rows, offset) and returned to the Client Application.

GTFS Translation API

This API returns the GTFS Feed resulting from translating the Urban Mobility NGSi counterparts. The result will include all the GTFS tables mapped from NGSi entities that have a relationship with the ones taken in input by the specific API. The following diagram depicts the expected interactions in the `getGTFSByAgency` case.

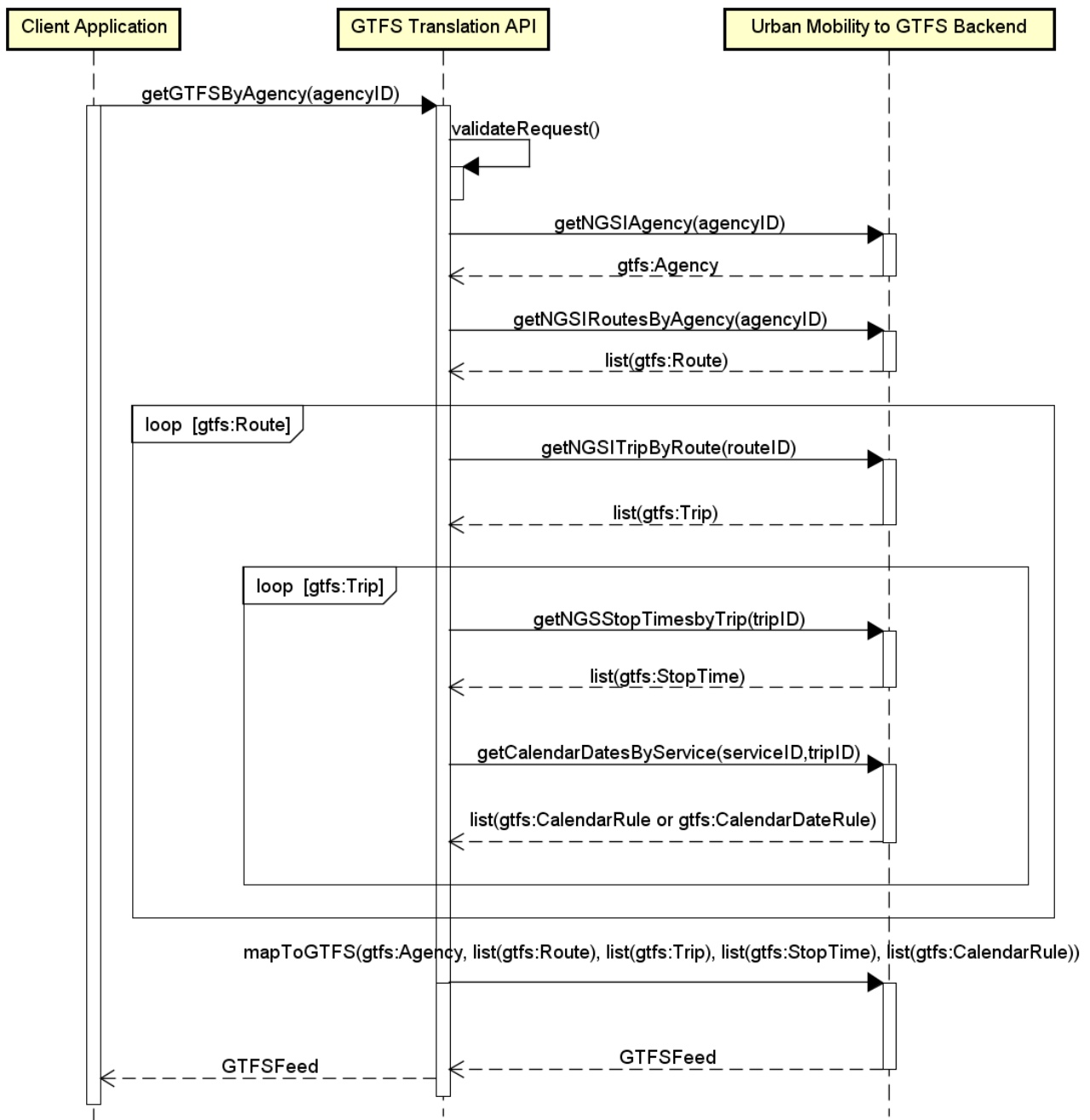


Figure 22 - `getGTFSByAgency` Sequence Diagram

The Client Application requests for a GTFSFeed, containing all the transportation schedules info issued by the Agency in input (agencyID). The Urban Mobility to GTFS API component validates the input request and forwards it to the Urban Mobility to GTFS Backend, which will retrieve all the available NGSI gtfs:Route entities for that agency. For each resulting route, the API component requests the list of its gtfs:Trip entities. For each resulting trip, the API component requests to the Backend the list of its gtfs:StopTime, which will retrieve the available entities together with their related gtfs:Stop (omitted for simplicity). Furthermore, the API component requests to the Backend the list of trip's gtfs:CalendarRule and gtfs:CalendarDateRule, by issuing the serviceID taken from its hasService field. Then, all the resulting entities are used by API component, to send to the Backend the request for the final GTFSFeed archive. The Backend maps every input NGSI entity to the corresponding GTFS member file and return the resulting Feed archive. Finally, the Urban Mobility to GTFS API component returns the resulting GTFS Feed archive to the Client Application.

In case of no input agencyID, interaction is slightly different: the operations are iterated for all the

available agencies and the final GTFS Feed will contain all the entities together.

Service Controller temporized job

The Service Controller performs a temporized job, in order to monitor updated NGSI entities that will result in an updated GTFS Feed archive file. The Figure 23 depicts interactions occurring among internal components of the Urban Mobility to GTFS service backend.

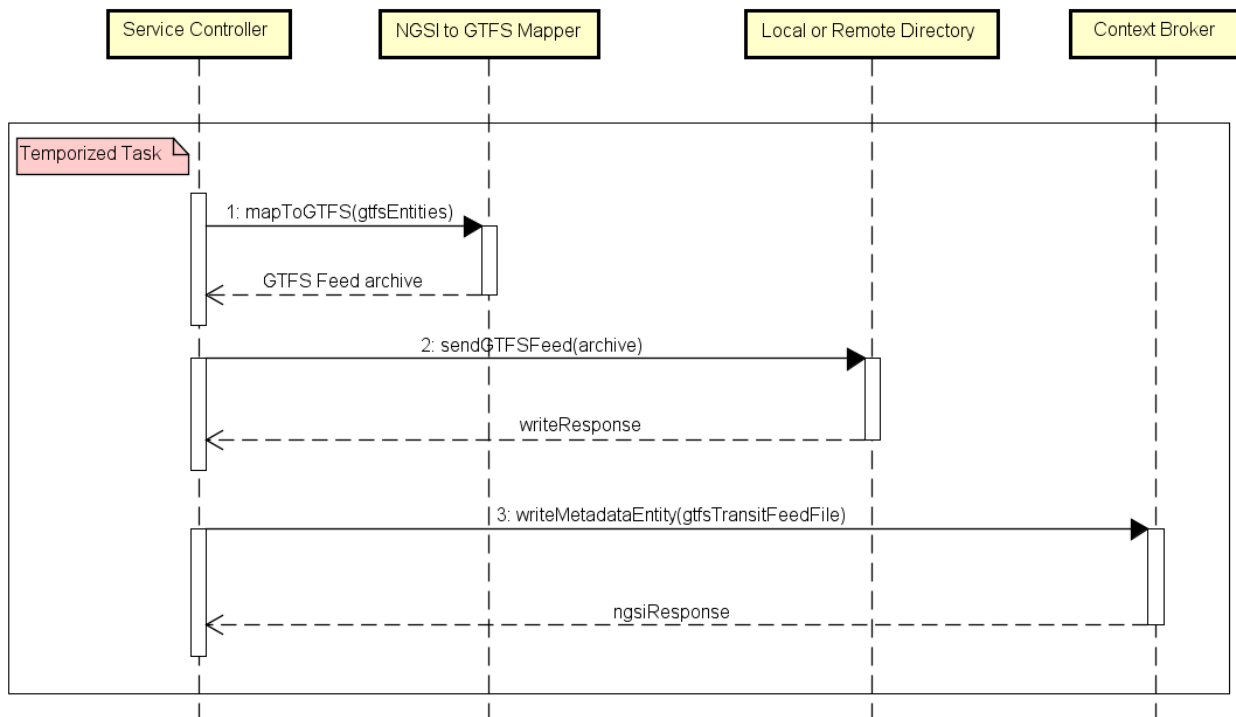


Figure 23 - GTFSFeed update job

The Service Controller, at a fixed time interval, requests the Urban Mobility Grabber for query needed NGSI entities (not included in this diagram for simplicity, as the NGSI retrieval operations are the same of Figure 22). The Service Controller compares resulting GTFS Feed archive file (through MD5 hash comparison) with the one already stored. In case the file has changed, it is sent to the Local or Remote Directory, where it is stored and made available publicly as a download URL. Then, the Service Controller fills a NGSI request with a `gtfsTransitFeedFile` entity [53], containing the metadata about this GTFS Feed, including the download URL where the stored archive file is available. Finally, the Service Controller sends the NGSI request directly to the Context Broker, in order to create or update the stored `gtfsTransitFeedFile` entity.

2.8.4 Service Architecture

This section describes the internal components that compose the service architecture and the logical workflow that occurs among them while processing the requests.

2.8.4.1 Service Components

As depicted in Figure 24, the service relies on SynchroniCity platform, in particular on the Context Broker and its NGSI APIs. It will also provide a RESTful API that exposes the service APIs.

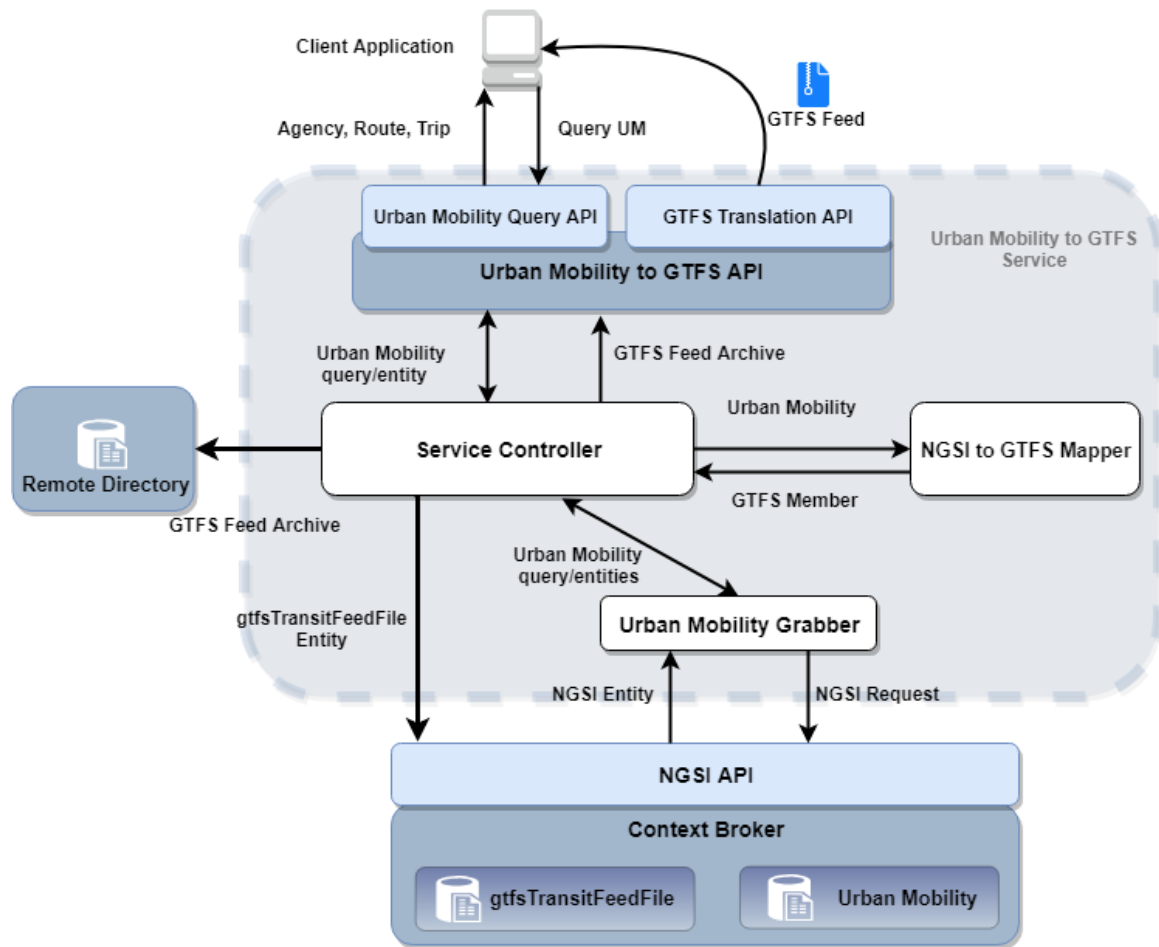


Figure 24 - Urban Mobility to GTFS Service architecture

The service is composed by the followings components:

- Urban Mobility to GTFS API:** this component represents the entry point of the service and exposes the functionalities provided by the service, implementing both **Urban Mobility Query API** (UM Query API) and **GTFS Translation API**, in order to expose their functionalities, described in §2.8.3.1. It realizes the API services through a RESTful standard interface and is responsible of validating the incoming requests, forwarding to other service components and finally returning appropriate payloads as response.
- Urban Mobility Grabber:** is the component that realizes the connection bridge between the service and the Context Broker through NGSI API. It is in charge of making the right NGSI requests through HTTP or HTTPS connection, in order to retrieve the Urban Mobility Context entities, which will be used either as Query result towards the upper UM Query API component or as input in the GTFS Feeds mapping process.
- NGSI to GTFS Mapper:** this component does the main work of the service itself, by implementing the logic to map each incoming NGSI entity to a GTFS Table member, namely the CSV file, which will be included in the final GTFS Feed archive. Thus, it performs a mapping from a JSON Object to a CSV file, according to the mapping rules identified by Urban Mobility specification [52]. It will be responsible also of validating;
 - Input Urban Mobility entities against their Data Model JSON Schemas, in order to ensure that input data are really compliant to one of the Urban Mobility data models.
 - Output GTFS Member with internal checking logic derived from GTFS Specification

[8], regarding mandatory fields and their semantic correctness.

- **Service Controller:** this is the core of the service and is responsible of following tasks:
 - o **Forwarding** requests among other components and performing specific logic for each incoming request. It will instruct the Urban Mobility Grabber component to retrieve, through remote NGSI requests, all the entities needed to perform the translation to the GTFS Feed. For instance, for a specific Agency, the Controller will ask for: all routes refereed to this Agency, all trips refereed to these routes, all the Stop Times and Stop related to each resulting Trip, all the CalendarRule for each Trip, and so on. Then, the Controller will interact also with the NGSI to GTFS Mapper, in order to translate all the collected entities into GTFS files. The results will be compressed in the final GTFS archive.
 - o **Storing** translated GTFS Feeds and keeping them updated by monitoring, through the Urban Mobility Grabber, the original NGSI entities present in the Context Broker. In detail, the monitoring task is realized through a temporized job that retrieves the entities, creates the GTFS Feed archive and compares its MD5 hash with the one already stored, in order to determine if the original NGSI entities and thus also the resulting archive file have changed. The GTFS Feed archive can be internally stored or, as depicted in Figure 24, sent to an external Remote Directory.
 - o **Creating** and updating the specific **GTFSTransitFeedFile** entity in the Context Broker. This entity provides metadata about the resulting GTFS Feed (stored either in a local or remote Directory), such as the URL where the resulting ZIP file is available to download. As described previously, this process is critical to realize the connection between this service and the GTFS Fetcher service (Section §2.7).

The interaction details for last two tasks are depicted in the Figure 31.

SynchroniCity platform components, external to the service:

- **Context Broker:** this component, belonging to the SynchroniCity architecture, is responsible of keeping in a persistent way the Context information as NGSI entities. The service will use the NGSIv2 API implementation provided by the FIWARE Orion Context Broker, in order to query and retrieve all those entities (e.g. Agency, Route, Trip, Stop Times, Calendar, etc.) needed for its functionalities.

2.8.4.2 Non-functional requirements

The following non-functional requirement is to be considered in the usage of the service:

- A RZ is supposed to take care of feeding the Context Broker with Context information, including those related to Urban Mobility entities. In addition, it shall make sure that the information sent to Context Broker is correct and consistent with relative Data Models.

2.8.5 Implementation Roadmap/Service deployment

2.8.5.1 Service implementation IT environment

The Urban Mobility to GTFS service is implemented as a backend web-based application running in a Docker [54] container environment. Once the docker image is instantiated, the atomic service will run either as command line tool or in server mode, interacting with the SynchroniCity infrastructure.

2.8.5.2 Service deployment dependencies

The Urban Mobility to GTFS service requires basically only one service from the SynchroniCity reference infrastructure:

- FIWARE *Orion Context Broker* hosts the NGSI Urban Mobility entities that will be retrieved by the service and translated to GTFS feeds.

Therefore, the service will need to be configured with the URL where the instantiated Context Broker is reachable.

2.8.5.3 Installation and configuration guide

The installation guide, describing how to get the dockerized service (docker image) and to correctly configure the application, is going to be published on the dockerhub page²⁸.

2.8.6 Support

Depending on usage of the service in particular application, user feedbacks or whenever GTFS specification will be updated, the service will be improved and eventually fixed. Therefore, the developing team will provide all the support²⁹ for Urban Mobility to GTFS service, which includes:

- Bug fixing
- Consultations about operation, use and exploitation of the service
- Installation, configuration and technical support in general.

Which do not include:

- The development of new functionalities not strictly related to the translation/mapping process from Urban Mobility to GTFS.

2.8.7 License & Terms and Conditions

The Urban Mobility to GTFS service is released under the terms of GNU Affero General Public License (version 3), The full document can be found at [55].

²⁸ <https://hub.docker.com/r/synchronicityiot/ngsiurbanmobility-to-gtfs>

²⁹ <https://gitlab.com/synchronicity-iot/ngsiurbanmobility-to-gtfs/issues>

2.9 GTFS-RT Loader from NGSI atomic service

This GTFS-RT Loader from NGSI is a brand new atomic service. This component is necessary in order to integrate the Routing service with the SynchroniCity framework.

2.9.1 Description

This service provides notifications of real-time information in a widely used format, GTFS-RT. In particular the atomic service consumes information following the SynchroniCity data models using an NSGI API, and generates GTFS-RT feeds from it. Initially the service has been developed to manage transportation arrival estimation, and it will be upgraded to also include other GTFS-RT kind of data.

The real-time entities stored in the SynchroniCity instances are bound to a GTFS feed file that contains the static data, what allows the service to support real-time provisioning for multiple GTFS feeds.

From a user perspective, the service is seen as a regular GTFS-RT provider. This way, legacy services able to consume GTFS-RT information can be seamlessly deployed on top of a SynchroniCity instance.

2.9.2 Application theme Functional Requirement addressed

This atomic service addressed the functional requirement FR-13 of the multi-modal transportation application theme. In particular, the service provides real-time feeds in a widely used format (GTFS-RT) generated from SynchroniCity data.

FR code	FR description	Comments
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	GTFS-RT Loader atomic service allows real-time data (such as air quality) announced via NGSI to be injected to Routing atomic service as input.
FR-13	The application may consider real-time information (deleted rides, delays, etc.) to better plan and update the journey	The GTFS-RT loader enables the final service to address this requirement.

Table 36 Multi-modal transportation application theme FRs addressed

2.9.3 Interface

The service interface is minimal and focuses on covering the functionality defined for GTFS-RT [9]. The service APIs are divided between those dedicated to the service management, and the ones used to consume the GTFS-RT data.

2.9.3.1 Management API

API Name		Add GTFS feed			
Description		This API is used to manage the service while it is running. In particular it allows to add more GTFS feeds for which the service must provide real-time information in GTFS-RT format. The query allows indicating the particular real-time feeds to provide. If it is not indicated, it will try to generate all.			
End-Point		/management/gtfsFeeds/{gtfsFeedName}?{tripUpdates, serviceAlerts, vehiclePositions}			
HTTP Method		POST			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
feedNames	Query	Array of string	No	tripUpdates, serviceAlerts, vehiclePositions	List of GTFS-RT types of update to generate
gtfsFeedName	URN	String	YES	n.a.	Name of the GTFS feed for which the GTF-RT is generated.
Responses					
Code		Content-type	Entity		Description
204		n.a.	n.a.		Operation finished successfully.
401		n.a.	n.a.		Unauthorized. The service is not authorized to get subscribed to the required entities.

Table 37 Add GTFS feed method

API Name		List GTFS feeds			
Description		The service will return the list of the GTFS feeds for which it provides real-time data, indicating the particular kind of GTFS-RT feeds provided.			
End-Point		/management/gtfsFeeds			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
Responses					
Code		Content-type	Entity		Description
200		application/json	FeedsInformation		List of GTFS-RT feeds that generate, along with statistical values of generation and usage

500	application/json	n.a.	Server error
-----	------------------	------	--------------

Table 38 List GTFS feeds method

2.9.3.2 Service API

API Name		Get GTFS-RT feed			
Description		Retrieve GTFS-RT data. The return of this query is a binary file to be consumed using proto buffer, as specified for GTFS-RT.			
End-Point		/{gtfsFeedName}/{gtfsRtUpdate}			
HTTP Method		GET			
Parameters					
Name	Type	Data Type	Mandatory	Default	Description
Responses					
Code		Content-type	Entity		Description
200		application/json	Binary GTFS-RT file		Regular use of GTFS-RT source
404		application/json	n.a.		Not found

Table 39 Get GTFS-RT feed method

Additionally the service provides a websocket interface to get subscribed to the real-time updates. The path used for the websocket clients is the same as the one to retrieve the updates.

2.9.4 Service Architecture

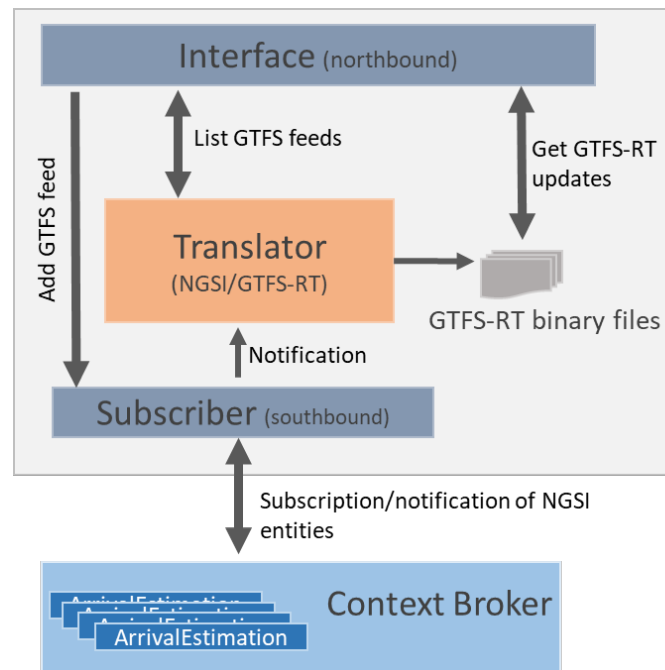


Figure 25 GTFS-RT loader from NGSI service architecture

- **Subscriber:** this component is in charge of creating the NGSI subscriptions to the corresponding entities in the Context Broker. In addition, it also receives the notifications and sends it up to the translator.
- **Translator:** this is the core of the service. The translator component receives NGSI notifications from the subscriber and processes them to obtain the data required to create the GTFS-RT.
- **Interface:** this component is the northbound of the service and allows users and managers to interact with it using the aforementioned APIs. Upon the query of a GTFS-RT update, the interface retrieves the binary file that is locally stored and sends it back to the requester.

2.9.5 Implementation Roadmap/Service deployment

Although the service is still under development, it has been defined a clear implementation and testing roadmap. The different components described in the previous section are being separately developed and tested. In particular, the following validation is being, or will be, performed for each module:

- Considering the optional attributes of the ArrivalEstimation data model, the subscriber has to be able to identify whether or not the notified entities have the required information to create the GTFS-RT information feeds. In this sense, those entities with partial information are filtered out before reaching the translator module.
- The Translator component is being tested to ensure that it is able to manage GTFS-RT messages belonging to multiple static GTFS feeds. In particular, it is being fed emulating notifications for different GTFS- feeds, and it is checked that the module creates and manages separate GTFS-RT binary files.
- Finally, the northbound interface will be validated along with the rest of components to ensure that the whole atomic service provides the expected functionality.

The atomic service will not have additional deployment dependencies apart from the Docker framework to simplify replicability. Concerning SynchroniCity services, the GTFS-RT loader requires subscriptions with a context broker. Access credentials to perform the subscription may be specified

in the service configuration.

In the future the service will be extended to support subscriptions to multiple context brokers, both through configuration and by extending the management API.

2.9.6 Support

User support is provided through the SynchroniCity code repository of the atomic service. The recommended way is to use issues³⁰ and merge request³¹ functionalities of corresponding repository.

2.9.7 License & Terms and Conditions

All the components and micro-services that shape this atomic service are Open Source, under the terms of the GNU General Public License as published by the Free Software Foundation (version 3).

For more information and details about the licensing, we encourage the reader to go and check the Synchronicity GitLab repo

³⁰ <https://gitlab.com/synchronicity-iot/gtfsrt-loader-from-nsgi/issues>

³¹ https://gitlab.com/synchronicity-iot/gtfsrt-loader-from-nsgi/merge_requests

2.10 Atomic Services synopsis

In this section we summarize the important links for each of the atomic services that come in hand for the interested users. Furthermore, since some of the atomic services have been already developed prior SynchroniCity, the Docker page, Git repository, API page and support platforms of those services are the legacy ones.

	Description	Links	License
Routing	Docker Hub	https://hub.docker.com/r/synchronicityiot/routing	LGPLv3
	Git Repository	https://gitlab.com/synchronicity-iot/routing-atomic-service	
	API	https://digitransit.fi/en/developers/apis/1-routing-api/	
	Support	https://gitlab.com/synchronicity-iot/routing-atomic-service/issues	
Parking Estimator	Docker Hub	https://hub.docker.com/r/synchronicityiot/parking-estimator	GPLv3
	Git Repository	https://gitlab.com/synchronicity-iot/parking-estimator	
	API	https://synchronicityparkingestimator.docs.apiary.io/#	
	Support	https://gitlab.com/synchronicity-iot/parking-estimator/issues	
Traffic Estimator	Docker Hub	https://hub.docker.com/r/synchronicityiot/traffic-flow-estimator	GPLv3
	Git Repository	https://gitlab.com/synchronicity-iot/traffic-flow-estimator	
	API	https://synchronicitytrafficservice.docs.apiary.io/#	
	Support	https://gitlab.com/synchronicity-iot/traffic-flow-estimator/issues	
Smartcities Dashboard	Docker Hub	https://hub.docker.com/r/synchronicityiot/smartcitiesdashboardservice	Private
	Git Repository	https://gitlab.com/synchronicity-iot/SmartCitiesDashboardService	

	API	n/a	
	Support	https://gitlab.com/synchronicity-iot/SmartCitiesDashboardService/issues	
Metrics Visualizer (Grafana)	Docker Hub	https://hub.docker.com/r/synchronicityiot/grafanaatomicservice	Private
	Git Repository	https://gitlab.com/synchronicity-iot/GrafanaAtomicService	
	API	n/a	
	Support	https://gitlab.com/synchronicity-iot/GrafanaAtomicService/issues	
GTFS Fetcher	Docker Hub	https://hub.docker.com/r/synchronicityiot/gtfs-fetcher	GPLv3
	Git Repository	https://gitlab.com/synchronicity-iot/gtfs-fetcher	
	API	n/a	
	Support	https://gitlab.com/synchronicity-iot/gtfs-fetcher/issues	
NGSI Urban Mobility to GTFS	Docker Hub	https://hub.docker.com/r/synchronicityiot/ngsiurbanmobility-to-gtfs	AGPLv3
	Git Repository	https://gitlab.com/synchronicity-iot/ngsiurbanmobility-to-gtfs	
	API	https://ngsiurbanmobilitytogtfsapi.docs.apiary.io/	
	Support	https://gitlab.com/synchronicity-iot/ngsiurbanmobility-to-gtfs/issues	
GTFS-RT loader from NGSI	Docker Hub	https://hub.docker.com/r/synchronicityiot/gtfsrt-loader-from-nsgi	GPLv3
	Git Repository	https://gitlab.com/synchronicity-iot/gtfsrt-loader-from-nsgi	
	API	n/a	
	Support	https://gitlab.com/synchronicity-iot/gtfsrt-loader-from-nsgi/issues https://gitlab.com/synchronicity-iot/gtfsrt-loader-from-nsgi/merge_requests	

Table 40 Summary of the important links of the atomic services suite

3 Conclusions and future developments

Presented suite of atomic services, their description, characteristics and implementation, concludes Task 3.2. Task 3.2 presented an effort to validate and explore the ability to co-design and co-create technical services, which can be leveraged by different RZs to accelerate adoption and reach effectiveness earlier based on common City Service requirements. This report follows an earlier publication called 'Suite of Atomic Services - Basic' [1] and it represents an evolution of the portfolio of technical atomic services. All Services presented here are horizontally supported by SynchroniCity partners and adopted by multiple cities, hence scaling city platforms without vendor or city lock-in. New services can leverage those atomic services gaining time-to-market and inheriting architectural robustness and implementation quality.

This establishes the technical foundations of a digital single market for IoT-enabled smart cities in Europe and beyond, thus targeting the Project Objective (PO) 1 of the SynchroniCity project. In fact, the atomic 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.

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 has been the input for the identification of new atomic services and the creation of new developing teams that led to a shared effort and co-creation of new atomic services enlarging the offer of IoT-enabled city services. The work described in this document evolves from the outcome presented in [1] and it reports implementations of designed services (i.e. Parking Probability Estimator and Traffic Flow Estimator atomic services), additions of new features to already selected atomic services (i.e. Metrics Visualizer atomic service), selection of proposed atomic services (i.e. GTFS-RT loader), and identification of new needs concretized into implementation of new atomic services (i.e. NGSI Urban Mobility to GTFS, and GTFS Fetcher atomic services).

The general Functional Requirements per application theme identified within Task 3.1 and described in [12] are covered in the SynchroniCity reference architecture [2] and the atomic services depicted in this document. As shown in Table 41, more the 70% of the FRs is addressed (see Annex for a full view).

	Common FRs	FRs Addressed	%
Human-centric management	8	7	87.5
Multi-modal transportation	28	21	75
Community Policy Suite	8	4	50
traffic	44	32	72.72

Table 41 Percentage of addressed FRs in common between RZs

The described atomic services in this document are at different maturity levels.

Beyond the end of Task 3.2, of which this document is the last report, we foresee a continuous evolution of atomic services that will address future needs and requirements of thirds parties, either

coming from the open call or external. With the completion of this task we wanted to demonstrate a viable way to co-creation and shared atomic services ecosystem for developing smart cities IoT applications, viz. city services. Therefore the roster of the atomic services is not stuck to the presented one but it is open to grow and advance when new necessities will appear and/or new contributor to this ecosystem will offer their expertise. For that reason we have opened a group on GitLab³² which act as a forum for the following purpose:

- Offer or requesting new features to existing atomic services
- Offer or requesting new atomic services

³² <https://gitlab.com/atomicservicesrevolution>

Bibliography

- [1] SynchroniCity, *D3.2 - Suite of baseline implementations - basic*, SynchroniCity, 2018.
- [2] SynchroniCity, *D2.1 Reference Architecture for IoT Enabled Smart Cities*, SynchroniCity, 2017.
- [3] OMA, *OMA NGSI-9/10*, OMA, 2012.
- [4] SynchroniCity, *SynchroniCity Historical API*, <https://synchronicityiot.docs.apiary.io/#reference/data-storage-api-historical/get-historical-data>, 2017.
- [5] *DCAT-AP : Data Catalogue vocabulary (DCAT) Application for data portals in Europe*, EU, 2015.
- [6] SynchroniCity, *D2.2 Guidelines for the definition of OASC Shared Data Models*, SynchroniCity, 2018.
- [7] SynchroniCity, *D3.5 Customized IoT service prototypes for lead ref. zones – basic*, SynchroniCity, 2018.
- [8] Google, "General Transit Feed Specification Reference," [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#field-definitions>.
- [9] Google, "GTFS Realtime Overview," [Online]. Available: <https://developers.google.com/transit/gtfs-realtime/>.
- [10] SynchroniCity, *D2.10 Reference Architecture for IoT Enabled Smart Cities, Update*, SynchroniCity, 2018.
- [11] SynchroniCity, *D3.4 Common methodology and toolset for city service customization*, SynchroniCity, 2018.
- [12] SynchroniCity, *D3.1 Documentation of service designs*, SynchroniCity, 2018.
- [13] SynchroniCity, "SynchroniCity GitLab repository," [Online]. Available: <https://gitlab.com/synchronicity-iot>.
- [14] SynchroniCity, *D3.6 Customized IoT service prototypes for lead ref. zones – advanced*, SynchroniCity, 2019.
- [15] OpenTripPlanner, *Open Trip Planner API (v0.15.0)*.
- [16] Facebook, *GraphQL*, 2018.
- [17] SynchroniCity, *D3.2*, SynchroniCity, 2018.
- [18] "SynchroniCity: Parking estimator API," 2018. [Online]. Available: <https://gitlab.com/synchronicity-iot/parking-estimator>.

- [19] "Synchronicity Parking Estimator API Documentation (Apiary)," 2018. [Online]. Available: <https://synchronicityparkingestimator.docs.apiary.io/>.
- [20] "Synchronicity Parking Estimator Docker Hub," 2018. [Online]. Available: <https://hub.docker.com/r/synchronicityiot/parking-estimator>.
- [21] "Synchronicity AI Prediction Module Docker Hub," 2018. [Online]. Available: <https://hub.docker.com/r/synchronicityiot/ai-prediction-engine>.
- [22] Keras, "Keras Webpage," 2018. [Online]. Available: <https://keras.io>.
- [23] "Tensorflow Website," 2018. [Online]. Available: <https://www.tensorflow.org/>.
- [24] S. Hochreiter and J. Schmidhuber, "LONG SHORT-TERM MEMORY," p. 32, 1997.
- [25] colah, "Understanding LSTMs," [Online]. Available: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [26] Microway, "Keras Theano Deep Learning Frameworks," [Online]. Available: <https://www.microway.com/hpc-tech-tips/keras-theano-deep-learning-frameworks/>.
- [27] "Synchronicity Traffic Flow Estimator API," 2018. [Online]. Available: <https://synchronicitytrafficflowestimator.docs.apiary.io>.
- [28] "Synchronicity Traffic Flow Estimator Docker," 2018. [Online]. Available: <https://hub.docker.com/r/synchronicityiot/traffic-flow-estimator>.
- [29] SynchroniCity Smartcities Dashboard Docker, "SynchroniCity Smartcities Dashboard Docker," [Online]. Available: <https://hub.docker.com/r/synchronicityiot/smartcitiesdashboardservice>.
- [30] SynchroniCity Metrics Visualizer Atomic Service, "SynchroniCity Metrics Visualizer Atomic Service," [Online]. Available: <https://hub.docker.com/r/synchronicityiot/grafanaatomicservice>.
- [31] Leaflet, "Leaflet providers preview," [Online]. Available: <https://leaflet-extras.github.io/leaflet-providers/preview/>.
- [32] F. Foundation, "Urban Mobility Data Models," 2018. [Online]. Available: <https://fiware-datamodels.readthedocs.io/en/latest/UrbanMobility/doc/introduction/index.html>.
- [33] SynchroniCity, "Urban Mobility Data Models," 2018. [Online]. Available: <https://gitlab.com/synchronicity-iot/synchronicity-data-models/tree/master/UrbanMobility>.
- [34] Google, "GTFS Static Overview," [Online]. Available: <https://developers.google.com/transit/gtfs/>.
- [35] "The Open Transport Working Group's github organization", "Linked GTFS specification," 2018. [Online]. Available: <https://github.com/OpenTransport/linked-gtfs/blob/master/spec.md>.
- [36] Google, "GTFS Reference - agency.txt," 2018. [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#agencytxt>.

- [37] Synchronicity, "Urban Mobility - Agency," 2018. [Online]. Available: <https://gitlab.com/synchronicity-iot/synchronicity-data-models/blob/master/UrbanMobility/Agency/doc/spec.md>.
- [38] Schema.org, "Postal Address," [Online]. Available: <https://schema.org/PostalAddress>.
- [39] Google, "GTFS Reference - routes.txt," 2018. [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#routetxt>.
- [40] Synchronicity, "Urban Mobility - Route," 2018. [Online]. Available: <https://gitlab.com/synchronicity-iot/synchronicity-data-models/blob/master/UrbanMobility/Route/doc/spec.md>.
- [41] Google, "GTFS Reference - trips.txt," 2018. [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#triptxt>.
- [42] Synchronicity, "Urban Mobility - Trip," 2018. [Online]. Available: <https://gitlab.com/synchronicity-iot/synchronicity-data-models/blob/master/UrbanMobility/Trip/doc/spec.md>.
- [43] Synchronicity, "Urban Mobility - Service," 2018. [Online]. Available: <https://gitlab.com/synchronicity-iot/synchronicity-data-models/blob/master/UrbanMobility/Service/doc/spec.md>.
- [44] Google, "GTFS Reference - stops.txt," 2018. [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#stopstxt>.
- [45] Google, "GTFS Reference - stop_times.txt," 2018. [Online]. Available: https://developers.google.com/transit/gtfs/reference/#stop_timestxt.
- [46] Google, "GTFS Reference - calendar.txt," 2018. [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#calendartxt>.
- [47] Google, "GTFS Reference - calendar_dates.txt," 2018. [Online]. Available: https://developers.google.com/transit/gtfs/reference/#calendar_datestxt.
- [48] Google, "GTFS Reference - fare_rules.txt," 2018. [Online]. Available: https://developers.google.com/transit/gtfs/reference/#fare_rulestxt.
- [49] Google, "GTFS Reference - shapes.txt," 2018. [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#shapetxt>.
- [50] Google, "GTFS Reference - frequencies.txt," 2018. [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#frequenciestxt>.
- [51] Google, "GTFS Reference - transfers.txt," 2018. [Online]. Available: <https://developers.google.com/transit/gtfs/reference/#transferstxt>.
- [52] SynchroniCity, "Urban Mobility - Introduction," 2018. [Online]. Available: <https://gitlab.com/synchronicity-iot/synchronicity-data-models/blob/master/UrbanMobility/doc/introduction.md>.

- [53] FIWARE, *FIWARE Urban Mobility data model*. Online: <https://fiware-datamodels.readthedocs.io/en/latest/UrbanMobility/doc/introduction/index.html>, FIWARE, 2018.
- [54] Docker, "Docker," [Online]. Available: <https://www.docker.com/>.
- [55] GNU, "GNU Affero General Public License," [Online]. Available: <https://www.gnu.org/licenses/agpl-3.0.en.html>.
- [56] "General Data Protection Regulation - Official Journal of the European Union, L 119," 4 May 2016.
- [57] SynchroniCity, *D3.3 Suite of baseline implementations - advanced*, SynchroniCity, 2018.
- [58] ETSI, *ETSI Context Information Management (CIM); Application Programming Interface (API): ETSI GS CIM v1.1.1*, ETSI, 2018.
- [59] SynchroniCity, "SynchroniCity DockerHub," [Online]. Available: <https://hub.docker.com/u/synchronicityiot/>.

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 service atomic
FR-10	The system should allow selection of specified time window and display corresponding aggregated data over the selected time window	4,33	Metrics (Grafana) service Visualizer atomic
FR-11	The system may provide advice/prediction based on aggregated historical data	3,67	Metrics (Grafana) service Visualizer atomic
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 service; Metrics Visualizer (Grafana) atomic 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 service atomic
FR-12	The system may allow for annotation specified time windows	2,67	Metrics (Grafana) service Visualizer atomic
FR-2	System should recognize preconfigured traffic situations and classify these situations in real-time	2,33	

Table 42 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 service atomic
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	Routing service + NGSI Urban Mobility to GTFS + GTFS Fetcher atomic
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	4,75	Routing service atomic

	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	4,75	Routing service + atomic GTFS-RT loader
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 service atomic
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	Routing service + atomic GTFS-RT Loader atomic service
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 service atomic
FR-16	The application must provide users with a georeferenced search of parking areas.	3,75	SynchroniCity reference infrastructure; Smartcities Dashboard atomic 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 atomic 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 and access control at least by a username/email address and password basic security (log-in).	3,00	SynchroniCity 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 atomic 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 atomic service; GTFS Fetcher; NGSI Urban Mobility to GTFS atomic service
FR-15	The application should be able to consider user related information to suggest routes based on his/her preferences.	2,75	Routing service atomic
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 atomic 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 43 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	4,67	Partially by SynchroniCity

	visualization		reference infrastructure; Metrics Visualizer (Grafana) atomic service
FR-4	UI requirements; system should be intuitive and easy to use	4,67	Smartcities Dashboard atomic service; Metrics Visualizer (Grafana) atomic 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) atomic 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 44 Community Policy Suite application theme FRs already addressed