Federation of Smart City Services via APIs

Pierfrancesco Bellini
DISIT Lab
University of Florence
Florence, Italy
pierfrancesco.bellini@unifi.it

Davide Nesi
DISIT Lab
University of Florence
Florence, Italy
davide.nesi@unifi.it

Paolo Nesi
DISIT Lab
University of Florence
Florence, Italy
paolo.nesi@unifi.it

Mirco Soderi
DISIT Lab
University of Florence
Florence, Italy
mirco.soderi@unifi.it

***Abstract*— In the context of Smart City, it is quite frequent the usage of Smart City API for providing services at web and mobile applications. Most of the solutions using Smart City APIs are focused on a single city. This means that passing from one city/area to another, the users must change application. This happens for the lack of interoperability among Smart City APIs and/or services. In this paper, the problem of federation of smart city services is addressed by proposing a solution for federating smart city APIs. To this end, a formal model has been proposed to federate API services, with efficiency, security, scalability, and capacity of managing overlapped areas of competence, distributed searches, etc.. These features are typically not all satisfied by classic GIS solutions which federate the services at level of databases. The solution has been developed in the context of Snap4City European platform enhancing former Km4City API of Sii-Mobility national project with Snap4City (https://www.snap4city.org).**

Keywords—knowledge base, smart city API, smart city services, federation of smart cities

I. INTRODUCTION

In the context of Smart Cities, not all cities/areas are becoming smart in the same manner [1]. In most cases, the cities are focused to a set of smart services, for example: smart parking, smart education, smart gov, smart lighting, etc., according to their needs and strategies. In most cases, vertical applications have been implemented for years as separate pillars, and only recently there is push on integrating data and/or services to exploit higher level machine leaning business intelligence tools and control room dashboards [2]. For Smart City applications, it is quite frequent the usage of Smart City API to provide services and data via web and mobile applications. Examples of Smart City APIs are: Km4City API [3], E015 [4], or [5]. Others possible clients could be control room Dashboards used by city operators, City Major and city Councilman [2]. Most of the solutions (web or mobile) using the Smart City API are focussed on a single city. This means that, passing from one city/area to another, the users have to change mobile application to get the same service. This happens for the lack of interoperability among Smart City API, SCAPI. They are not standardized, see the review of Smart City API on [6]. An alternative solution is offered by global services (such as Google) which sadly not covers local services using private data of the city. A large part of the services proposed via Smart City API are geolocalized and provide different results according to the user's ID assigned profile: organization, role level, past activity, preferences, etc. The User ID implies the management of Personal Data, and thus the GDPR has to be applied [7]. In the context of GIS (Geographic Information Systems) data exchange the federation of services is largely diffused [8]. Most of the GIS solution for federation are based on federating datastores. The classic GIS interoperability is limited to 1:1 exchange of geographical data for example exploiting protocols as WFS (Web Feature Service), WMS (Web Map Service), for the exchange of Maps and geoelements such as paths, points of interest, road elements, road graphs, etc. GIS protocols, according to their definition, they are typically unsuitable as API for providing data related to smart services, such as smart parking, subscription on alerts on environmental conditions, etc., as needed for smart city applications. Most of vertical Smart City API based solutions are typically focussed on a limited range of data by using SQL databases which in turn can provide support for geolocated information and may be federated at level of the database. In alternative, noSQL storages can be designed to support the smart city services as well, also supporting geolocated information. Among the noSQL storage solutions, the most suitable to manage geolocated information together with city entities relationships are those based on RDF model (Resource Description Framework) [9]. RDF stores can be federated at level of semantic model. On the other hand, federated RDF storages are far to be a good support for federated smart city services [10], since the concepts of federation has to be at service level, for example, for routing by taking into account different geographical areas addressed by different RDF stores.

In this paper, a solution for federating Smart City API on geographic area is presented. The study has been motivated by scaling Km4City Smart City API from single cities to regions and from regions to a number of European Areas in the context of Snap4City PCP for Helsinki, and Antwerp [7]. The solution proposed aimed to solve the requirements that presently cannot be solved by traditional GIS solutions. The solution proposed does not need to: (i) create a strong cohesion among the different services and thus does not leave at the single city the decision to join the federation or not and neither to the single client Mobile application, (ii) connect GIS services via WFS/WMS services, (iii) migrate data among services. For the design, implementation and validation of the solution we have used and enhanced the former Km4City API and ontology [6], which has been implemented as the so called ServiceMap. For this reason, the federation of Smart City API has been provided in the so-called SuperServiceMap. The validation has been performed by considering 4 large areas and smart city services in place covering: Tuscany region of 3.5 inhabitant in the center of Italy, north of Italy and Sardegna island, Antwerp and north of Belgium, and Helsinki and south of Finland.

The paper is organized as follows. In section II, the requirements identified for federating Smart City API are presented. Section III presents the general architecture and solution for federating large smart city services, which may cover single cities, as well as regions. In Section IV, a formal operational model and semantic for the federation of smart city API and services are presented. Section V describes the main families of the Advanced Smart City API which can be exploited in the federated network. Section VI provides the evidence of how the identified requirements identified in Section II have been satisfied. Validation and performance have been reported in Section VII. Conclusions are drawn in Section VIII.

978-1-7281-6997-2/20/$31.00 ©2020 IEEE
DOI 10.1109/SMARTCOMP50058.2020.00077

II. REQUIREMENTS AND ANALYSIS

A distributed Smart City API, **SCAPI**, based solution for a set of cities, let say a federated network of Smart City APIs (called in the following federated **SCAPI)** should satisfy the following requirements. The federated **SCAPI network** connects a set of SCAPI services (called Nodes) which are independently offered and maintained by single cities/areas. They provide their own data and services. It should not be confused with the federation/collection of APIs in a common basket to expose them uniformly in terms of definitions but managed by offered by different organizations providing this service to a single city/area mainly as in E015 [4]. A more general view is that of ALMANAC [13], [11], which can be regarded as E015 with a limited scope in terms of services, among different cities. Thus, according to our definition the federated **SCAPI** network of nodes should:

1) guaranteed that **nodes do not need to permanently share data** as in P2P solutions or as in federated databases. This requirement must be satisfied to assure that the data are located only in the nodes authorized to manage them;
2) support **distributed search** on the federated **SCAPI** network;
3) support **nodes of any size** in terms of number and volume of data sets providing services of the nodes. In addition, the geospatial size and shape of each node may be: (i) not regular (nor a circle but a shape), and multiple connected (so called multi-polygon), (ii) partially overlapped with other nodes, (iii) totally included into those of other nodes, (iv) disjoined and even far each other (this means that the union of all the areas can be disjoined with respect to the global map of the earth);
4) Support nodes with a **different number of services available**. This implies that not all kinds of services and data may be necessarily available in all nodes;
5) Support nodes with **georeferenced services or not.** This means that are general for the area addressed and not specifically related to the GPS position;
6) respond to API calls in terms of services in **transparent manner passing from one node to another** or when the service needs to provide results coming from more nodes;
7) support access control to **prevent access to data and services by not authorised users**. Since the passage of a user from one SCAPI node to another of the federated SCAPI network may imply the sending of requests which may try to access at private data/services;
8) support the **addition/removal of nodes in the network** without the need of fully restructuring of the network and modifications have an immediate effect without any service reloading or disruption;
9) provide **results in real time also when a large number of nodes/areas are involved**. The implementation should also provide support for creating redundant solutions with high resilience;
10) provide the **response in the coherent format** with the expected response of the single services. Thus, the results of the federation may need to be merged to produce the response in any format: JSON, XML or HTML.

III. GENERAL ARCHITECTURE

In order to satisfy the requirements, the **architecture reported in** Fig. 1 has been realized. It includes a Master of the federated SCAPI network implemented as Smart City Servers with SCAPI (in Snap4City they are the **ServiceMap** tools [12]). In order to avoid having a single point of failure, the Master can be replicated into each node and the list of Super services on top of **ServiceMap SCAPI** can be put accessible in one or more web servers for update. Each **ServiceMap** has a representation of the multi-polygon addressed by the nodes (with their data/services) and thus of their partitioning over the nodes of the federated SCAPI network. In more details, each node/SCAPI may register in the Master and Super network the descriptor of the multi-polygon area of your competence. This approach permits at the **Supers** to redirect the queries to the nodes that could provide the service and data. Thus the **Supers** do not need to hold the data of the nodes and perform the distribution of queries only to the involved nodes, to finally collect the results and performs data fusion. The **Supers as well as the ServiceMaps** may also implement some query caching solution as all the other SCAPI.

The most relevant aspects of the procedure are mainly related to the: (i) identification of nodes to be involved according to the requests received with a complexity that may depend on the size of the node descriptors (number of polygons); (ii) distribution of query in each **Super** with a complexity O(1), the execution time would be top to the slowest node, exploiting multithreading approach; (iii) collection of results and their fusion to avoid duplicates and compounding the results, with a complexity depending on the size S of the elements O(S). The main complexity is typically due to point (iii). When the nodes are created to split the complexity of computing services and thus to improve the performance: the workload on nodes has to be balanced. Moreover, in certain cases of (i), it is not possible to identify a priori one and only one node to be queried for a specific data/service, and may imply that in some cases all nodes are queried.

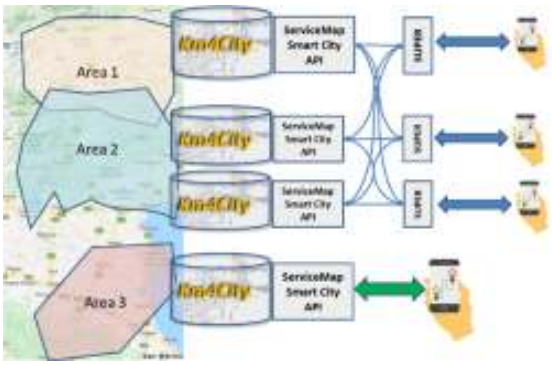

Fig. 1 – General Architecture of Super ServiceMap and Federated SCAPI network

IV. SUPERSERVICEMAP OPERATIONAL SEMANTIC

In this section, the ServiceMap model is presented and it is shown how different models are combined to create a **SuperServiceMap** exploiting former SCAPI based on Km4City plus some extension. In the context of the SuperServiceMap solution, a ServiceMap is defined by the following tuple:

$$ServiceMap = \langle E, G, A, D, loc, gs, ss, inf, path \rangle$$

Where:

- E is the set of city/IOT entities managed by the ServiceMap. The city entities can be POI, services, IOT devices, road segments, etc.;
- $G \subset W = [-180,180] \times [-90,90]$ is a subset of the WGS84 geographic coordinates representing the geographic area covered by the ServiceMap;
- A is the n-dim space that models the arguments that can be provided for a search request;
- D is the m-dim space that models the data associated with an entity (e.g., name, address, category);
- $loc: E \rightarrow \wp(G)$1 is a function mapping for each entity a geographic point or area in G;
- $gs: \wp(W) \times A \rightarrow \wp(E)$ is a function that models geographic search that given a subset of W and some other arguments in A returns a subset of entities E that belong to the geographic area so that $\forall X \subset W \ \wedge \ e \in gs(X, a) \rightarrow loc(e) \cap X \neq \emptyset$;
- $ss: A \rightarrow \wp(E)$ is a function modelling searches that do not involve geographic info, so that given the search parameters returns a subset of the entities satisfying the search;
- $inf: E \rightarrow D$ is a function which returns the list of details of a city entity;
- $path: G \times G \times A \rightarrow Seq(E)$ is a function that given two geographic points and other parameters returns a sequence of entities in E representing the "best" path/route (modal or multimodal, with different travel means).

This model allows to have entities with no geographic position i.e., $loc(e) = \emptyset$, in this case the gs function will never return these entities while they can be retrieved using the ss function. An important property of function gs is that if $X \cap G = \emptyset$, $gs(X, a) = \emptyset$ that can be read: searching outside the geographic coverage area of a ServiceMap will not give results.

Given a set of ServiceMaps $\{SM_1, SM_2 \ldots SM_N\}$ we can build the **SuperServiceMap** to manage all entities in $ServiceMap_i = \langle E_i, G_i, A, D, loc_i, sg_i, ss_i, inf_i \rangle$ defining:

- $E = \bigcup_{i=1}^{N} E_i$
- $G = \bigcup_{i=1}^{N} G_i$
- $loc = \bigcup_{i=1}^{N} loc_i$
- $gs(X, a) = \bigcup_{i=1}^{n} gs_i(X, a)$
- $ss(a) = \bigcup_{i=1}^{n} ss_i(a)$
- $inf = \bigcup_{i=1}^{N} inf_i$
- $path = \bigcup_{i=1}^{N} path_i$

The sets A of possible arguments and D of possible details are not tied to any particular ServiceMap. No coherence problems arise if the set of city entities of different ServiceMaps are disjoined ($\forall i \neq j E_i \cap E_j = \emptyset$). While, if some common city entities exists -- i.e., $\exists e \in E_i \wedge e \in E_j$ it has to hold: $loc_i(e) = loc_j(e)$ and $inf_i(e) = inf_j(e)$.

[1] $\wp(X)$ is the powerset of X, set of all possible subsets of X

Moreover, it should be noted that function $path$ of **SuperServiceMap,** defined as the union of $path_i$ functions over the different G_i, does not associate a value when applied to two points belonging to disjoint G sets. The error management allows to split the problems in distinct routing queries at the corresponding ServiceMap services.

V. ADVANCED SMART CITY API, ASCAPI

This section is devoted to the description of the Advanced SCAPI, **ASCAPI**. The full list and semantic of input arguments for API (space A) can be retrieved from the online documentation of the ASCAPIs on https://www.km4city.org/swagger/external/index.html. They have been strongly extended with respect to the former Km4City Smart City API of [Nesi et al., 2016]. The extensions have been designed and performed to extend the number of data types, domains and city entities addressed, and to allow the federation of SCAPI based knowledge bases.

The *Service discovery and information* family of APIs allow to perform a search for services (i) around a given position, (ii) near to a given service, (iii) within the boundaries of a bounding box, (iv) within the boundaries of a given geographic area (which can be defined as a WKT shape), (v) within the boundaries of a given municipality, (vi) through one of the above filtered on the basis of a full-text to be searched. The same API allows to retrieve full details about a given service of interest, identified through its URI (services are stored as resources in semantic knowledge bases, ServiceMaps -- also said the triplestores -- each identified through a URI, that also is a URL, in a Linked Data). This is modelled with the *inf* function that is used to retrieve information about an entity. If it is not possible to recognize from the URI the ServiceMap that is managing the referred entity the only choice is asking to all ServiceMaps and caching the (service, ServiceMap id) pair for future requests. This is performed in the middle layer as described in the following.

The *Event Search* family of APIs allows to retrieve events in the region (past, current and future). The geographical region of interest is specified by the user through the *selection* input parameter.

The *Address and geometry search by GPS* family of APIs allows to retrieve a full address (municipality, road name, and civic number) that corresponds to the nearest to a given position, also possibly including public transport lines or other points of interest that locate nearby the selected position, possibly furtherly filtered through a full-text search.

The *Search paths in a geographic area* family of APIs allows to retrieve paths of the public transports that locate nearby a geographical position or within the boundaries of a geographical area. In this case, search for all the bus-stops in the area supplied and returns all the bus lines that pass from those stops. The key input argument for the identification of the POI is the *selection*, modelled as *gs* function. The *Shortest Route/Path Finder* API allows users to get the best route from a given starting point to a destination through a modal or multimodal routing. In this case, it is identified the ServiceMap managing them (via cache or via request) and if exists a ServiceMap that match for both the source and destination it is queried otherwise an error is returned. The management of the error allows to split the problems in distinct routing queries at the corresponding ServiceMap

services.

VI. THE SUPERSERVICEMAP MAIN PROCESSING ASPECTS

In this section, the details about the SuperServiceMap is implemented according to the general architecture described in Section III. The ServiceMap exposing SCAPI are implemented in Java and RDF triple stores. The SCAPI provides results to the SuperServiceMap or directly to the clients by using JSON. Each ServiceMap may have or not its own SuperServiceMap interface and in that case may be connected to all the other ServiceMap as depicted in Fig. 1. The SuperServiceMap implements ASCAPI and the logic reported in Section IV, it: (i) identifies and query the ServiceMaps on the basis of the received requested and their georeferences, (ii) submits the queries to one or more ServiceMaps and get the results, (iii) combines the results and provides them to clients. Thus, when multiple ServiceMaps are queried, the request is delivered in parallel to all of them by the SuperServiceMap. Then the responses are collected and merged, duplicates are stripped out, resources are sorted with the same criterion that the *ServiceMaps* use, and the final response is provided back to the end user.

A. Selection of ServiceMaps to be queried and exceptions

The selection of the ServiceMap(s) to be queried is generally based on the geographic areas involved in the request. Thus, only the ServiceMap addressing/intersecting that area are queried. The pair (service URI, *Service Map*) is then cached to reduce the number of requests from one service to another. In the case of multiple ServiceMaps insisting on the same area. In order to maximize performance, or making an a priori balance on services, it has to be possible to define a priority among the different ServiceMaps on the basis of the query or on services. This mechanism is practically used to satisfy Reqs.5. The passage from one SuperServiceMap to another can be also very easy since the Mobile Application can have a direct link with the SuperServiceMap and area for which has been natively developed. While discovering that all the results are referring to another are may decide to continue to ask at its API end point of reference or to pass at the other one.

B. Merging ServiceMap results of any Format

ServiceMap ASCAPI (as smart city API) may be invoked to request results in JSON or HTML [Nesi et al., 2016]. Results in JSON can be easily merged, while results in HTML may include interaction plus relative URLs to JavaScript, style sheets, images, and other can be found. This approach is very suitable for mobile client which need to immediately provide the results without reformatting. Thus, for HTML, the SuperServiceMap needs to combine multiple results coming from different ServiceMaps which provide different base URLs. Thus, the HTML results received from the ServiceMaps are parsed, relative URLs are identified, and are replaced with full URLs.

The above subsections have discussed the solution to satisfy some of the above-mentioned requirements. Req.1 is satisfied by the solution since the different ServiceMap/ SuperServiceMap servers do not copy the data of the other services in local but only know the presence of the other services and have their high-level descriptor in terms of area of competence. Reqs. 2, 3, 4, 8 and 9 are satisfied and performance are shown in Section VII. Req.7 is satisfied according to security aspects addressed at level of single ASCAPI as described in [Badii et al., 2020]. Req.10 is described in Section VI.B. Reqs. 5 and 6 are discussed in Section VI.A.

VII. VALIDATION AND EXPERIMENTAL RESULTS

The solution has been validated, and we have also studied the performance in 4 different real operative conditions. To test the efficiency and effectiveness of the proposed approach, the following conditions have been set up: (i) three areas of interest with their ServiceMap have been taken: (1) Tuscany Region (about 519 Million of triples), (2) City of Antwerp and City of Helsinki (about 203 Millions of triples) including two disjoined areas, and (3) Italy regions of Sardinia, Emilia, Veneto, Lombardia and Santiago de Compostela (about 321 Millions of Triples, with a number of disjoined areas) (the data and areas are accessible from https://www.snap4city.org) (see Fig. 2 that presents the overlap between the two different KBs API for Tuscany and Emilia of Italy); (ii) a heterogeneous set of ASCAPI requests has been identified including geo and non geo based requests; (iii) five different solutions have been considered (see Table I), and described as follows:

- FSSM: the SuperServiceMap is forced to forward requests to all ServiceMaps (three ServiceMap VMs each of which with 16 GByte of RAM and 12 cores) any query is receiving from clients and merges the results;
- PSSM: the SuperServiceMap performs a selection of the most suitable ServiceMaps (among the three ServiceMap VM each of which with 16 GByte of RAM and 12 cores) to be involved on the basis of the geoinfo included in the query, and merges the received results;
- ASM: the client is sending the query directly to the ServiceMap of its referred area (a ServiceMap with 16 GByte of RAM and 12 cores);
- GSM: the client queries a ServiceMap in which all the data (triples) of the three areas have been stored (a VM with 16 GByte of RAM and 12 cores);
- PGSM: as GSM but with a ServiceMap VM with 48 GByte of RAM and 36 cores.

Measures for FSSM, PSSM and ASM have been performed by querying the real operative services in the corresponding cities and thus with several thousands of users were performing access at the same time on the services. GSM and PGSM have been autonomous servers isolated from the actual services. And thus, they can be taken as best case and not as a reference for the operative conditions. In fact, in some conditions the global solutions do not provide the requested results.

Thus, a number of clients have been set up to stress the configurations sending 100 times the same query and measuring the typical response time as reported in Table 1. The **ASM** column reports the performance in the case of direct query to the correct ServiceMap. **FSSM** could be considered a worst case, since no strategies for distribution are applied.

Fig. 2 – ServiceMaps borders overlapping in Italy (left), ServiceMap Areas (right), in Europe.

The **PSSM** solution has obtained a mean performance improvement of the 28% with respect to the FSSM, by taking into account all the queries. The **ASM** solution based on direct query still performs in average the 22,8% better than the PSSM solution. GSM solution has been demonstrated too be under dimensioned in terms of resources to provide interesting results, it provides an mean error of 45% worst with respect to the ASM with a variance in the order of 10^7. The solution **PGSM** seems to be the better ranked when the queries are simple as in Queries of kind: B, C, D. On the other hand, in most of the cases, the performance obtained by the PGSM solution is not drastically better than the PSSM which provides performance of the same order in most of the queries. Moreover, the PSSM solution provides results also in queries J in which the GSM and PGSM are not capable to provide results to the users providing an error code after 10 minutes (marked as * in Table 1). In those cases, the advantage of using the distributed solution **PSSM** is evident with respect to the single VM solutions PGSM. Moreover, the PGSM solution, despite to its better performance for simple queries, presents a very large variance with respect to the ASM reference solution in the order of 10^8, due to queries that does not produce acceptable results for typical geospatial queries such as those of type: A, which are requesting services near to a point. **PGSM** has been found the most efficient solution for simple queries (queries to which the PSSM also provides a response in a fraction of second), since it does not pay the network overhead of asking to the *ServiceMaps*, and waiting for responses. This means that the SuperServiceMap may implements more sophisticated strategies to improve the service performance in specific cases. In fact, the solution has also put in place a rule-based module to change the strategies according to the services, area, and query shape.

A remarkable test bed for the proposed approach also is the case in which the geographical region of interest falls across the boundary between two ServiceMaps. In those cases, the SuperServiceMap using PSSM identifies the specific ServiceMaps to be queried and *carefully* merges responses, composing a response: removing duplicates that could be present when ServiceMaps are partially overlap. As an example, you can query the SuperServiceMap based on PSSM for all services which are located along the path outlined in Fig. 3, that joints cities of Firenze and Bologna (approximately 80Km) tracing the path of the Highway A1, E35. The query browsing to the following address:

http://www.disit.org/superservicemap/api/v1/?selection=wkt:LINESTRING(11.248284683089423%2043.77705897010958,11.156274185042548%2043.82958894455573,11.259271011214423%2044.48868359713514,11.338648304801154%2044.49415965702086) producing a limit of 100 results of the 697 POI.

TABLE I. VALIDATION PERFORMED IN TERMS OF TESTED QUERIES ON THE FEDERATED NETWORK (IN BOLD THE BEST TIMES AND HIGHLIGHTED THE BEST TIMES BETWEEN PSSM AND PGSM)

Query/Kind		Times of Response (ms)			
		FSSM	***PSSM***	***ASM***	***GSM PGSM***
Get all services in a radius of 500 m from the center of Florence	A	5076	4670	4033	139329 132306
Get all services in a radius of 500 m from the center of Antwerp	A	1670	1474	1485	81161 76720
Get all services in a radius of 500 m from the center of Helsinki	A	2377	2148	1622	171713 160890
Get all events in a radius of 20 km from Florence this month	B	1,762	906	621	142 124
Get all events in a radius of 20 km from the center of Antwerp this month	B	1124	531	478	696 697
Get all events in a radius of 20 km from the center of Helsinki this month	B	656	525	404	179 131
Locate a given bar in the Municipality of Florence	C	2051	1572	1456	3259 761
Locate a given theatre in the Municipality of Antwerp	C	1581	1520	1404	783 703
Locate a given hotel in the Municipality of Helsinki	C	1272	1261	1086	908 599
Get full details about a given bar in Florence	D	535	314	196	1257 120
Get full details about a given theatre in Antwerp	D	327	233	132	206 94
Get full details about a given hotel in Helsinki	D	325	229	106	198 146
Locate all bars in a radius of 1 km from a given bar in Florence	E	4755	4671	3976	6725 5745
Locate all theatres in a radius of 1 km from a given theatre in Antwerp	E	1689	977	279	983 696
Locate all hotels in a radius of 1 km from a given hotel in Helsinki	E	1627	1234	285	982 841
Locate all cinemas in Florence or in its immediate nearby	F	307	242	185	275 202
Locate all healthcare centers in Antwerp or in its immediate nearby	F	449	337	264	207 163
Locate all pre-primary schools in Helsinki or in its immediate nearby	F	837	748	588	171 56
Get all events in Florence or in its immediate nearby this month	G	155	154	162	235 62
Get all events in Antwerp or in its immediate nearby this month	G	939	187	124	271 56
Get all events in Helsinki or in its immediate nearby this month	G	920	178	111	159 72
Locate restaurants in the district of Katajanokka, in Helsinki	H	539	351	266	919 365
Locate public transport stops in the small district of Borgerhout, in Antwerp	H	1118	1000	961	363071 342253
Locate a footwear shop in Rifredi (Florence)	H	506	247	230	198 178

Query/Kind		Times of Response (ms)			
		FSSM	*PSSM*	*ASM*	*GSM PGSM*
Get all events in Rifredi (Florence) this month	I	392	126	82	57 50
Get all events in Katajanokka, in Helsinki, this month	I	623	190	114	397 366
Get all events in the small district of Borgerhout, in Antwerp, this month	I	156	128	118	76 76
Get public transport routes that traverse the district of Rifredi (Florence)	J	1948	907	878	* *
Get public transport routes that traverse the district of Katajanokka Helsinki	J	4100	655	533	* *
Get public transport routes that traverse the district of Borgerhout, in Antwerp	J	1591	1377	1222	* *
Get the full address of the center of Florence	K	583	376	288	1002 355
Get the full address of the center of Antwerp	K	775	769	668	641 656
Get the full address of the center of Helsinki	K	799	757	588	778 768
Get public transport agencies that operate within 5 km from the center of Florence	L	141	121	70	319 206
Get public transport agencies that operate within 5 km from the center of Antwerp	L	183	149	149	54 35
Get public transport agencies that operate within 5 km from the center of Helsinki	L	121	104	73	61 35
Find the shortest path from the center to the airport in Florence	M	2040	1983	1916	2746 1977
Find the shortest path from the center to the airport in Antwerp	M	605	449	352	223 157
Find the shortest path from the center to the airport in Helsinki	M	375	366	338	243 223
Get all kind of services on a Linestring crossing the border on Tuscany and Emilia Romagna	N	2117	1919	-	NA 2004

Indeed, data about the Tuscany region are stored in a dedicated ServiceMap, while data about all other covered areas of Italy are stored in another. For such a query, the PSSM provides back the list of services in 1919ms while querying the PGSM it takes 2004ms.

VIII. CONCLUSION

This paper presented a solution for federating Smart City API. The solution aims to solve the requirements that presently cannot be solved by traditional GIS solutions. Thus we would avoid migrating data, providing federation at level of APIs, involving nodes of any size, combining them autonomously so that each of them may decide to join or not, leaving the possibility of having different kind of services, enabling the movements from among federate areas, prevent the access and respect GDPR and data security, combining services, etc. For the design, implementation and validation

Fig. 3 – A path crossing 2 ServiceMap areas.

of the solution we have used and enhanced the former Km4City API and ontology [Nesi et al., 2016], which has been implemented as the so called ServiceMap. For this reason, the federation of Smart City API has been provided in the so-called SuperServiceMap. The validation has been performed by considering 4 large areas and smart city services in place covering: Tuscany region of 3.5 inhabitant in the center of Italy, north of Italy and Sardegna island, Antwerp and north of Belgium, and Helsinki and south of Finland. The validation has shown that the solution performs better that single centralized services in many cases, except for the cases in which simple direct queries are performed. This means that the SuperServiceMap may implements a number of more sophisticated strategies to improve the service performance in specific cases. In fact, the solution has also put in place a rule-based module to change the strategies according to the services, area, and query shape.

REFERENCES

[1] Hernández-Muñoz, José M., et al. "Smart cities at the forefront of the future internet." The future internet assembly. Springer, Berlin, Heidelberg, 2011.

[2] P. Bellini, D. Cenni, M. Marazzini, N. Mitolo, P. Nesi, M. Paolucci, "Smart City Control Room Dashboards: Big Data Infrastructure, from data to decision support", Journal of Visual Languages and Computing, 10.18293/VLSS2018-030

[3] Badii, Claudio, et al. "Analysis and assessment of a knowledge based smart city architecture providing service APIs." Future Generation Computer Systems 75 (2017): 14-29.

[4] Zuccalà, Maurilio, and Emiliano Sergio Verga. "Enabling energy smart cities through urban sharing ecosystems." Energy Procedia 111 (2017): 826-835.

[5] Krylovskiy, A., M. Jahn, and E. Patti. "Designing a smart city internet of things platform with microservice architecture." 3rd Int.Conf. on Future Internet of Things and Cloud. IEEE, 2015.

[6] Nesi, Paolo, et al. "Km4City Smart City API: an integrated support for mobility services." 2016 IEEE International Conference on Smart Computing (SMARTCOMP). IEEE, 2016.

[7] C. Badii, P. Bellini, A. Difino, P. Nesi, "Smart City IoT Platform Respecting GDPR Privacy and Security Aspects", IEEE Access, 2020. 10.1109/ACCESS.2020.2968741

[8] Cinquini, Luca, et al. "The Earth System Grid Federation: An open infrastructure for access to distributed geospatial data." Future Generation Computer Systems 36 (2014): 400-417.

[9] Gyrard, Amelie, and Martin Serrano. "Connected smart cities: Interoperability with seg 3.0 for the internet of things." 2016 30th Int. Conf. on Adv. Inf. Networking and Applications (WAINA). IEEE.

[10] Makris, Konstantinos, et al. "Ontology mapping and SPARQL rewriting for querying federated RDF data sources." OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". Springer, Berlin, Heidelberg, 2010.

[11] Bonino, Dario, et al. "Almanac: Internet of things for smart cities." 2015 3rd International Conference on Future Internet of Things and Cloud. IEEE, 2015.

[12] Bellini, Pierfrancesco, et al. "Km4City ontology building vs data harvesting and cleaning for smart-city services." Journal of Visual Languages & Computing 25.6 (2014): 827-839.

[13] Soto, José Ángel Carvajal, et al. "Towards a Federation of Smart City Services." International Conference on Recent Advances in Computer Systems. Atlantis Press, 2015.