A Smart City Development kit for designing Web and Mobile Apps

C. Badii, P.Bellini, P. Nesi, M. Paolucci
University of Florence, Department of Information Engineering,
DISIT Lab, http://www.disit.org , http://www.sii-mobility.org , <name.surname>@unifi.it

***Abstract*—**Smart City services' effectiveness is enabled by the integration and availability of data coming from city operators on different domains: mobility, energy, health, water, telecom, tourism, culture, etc. They may be open and private data, static and real time. The most cases, smart city developers still have to develop their applications by studying several data sets and API sources, recovering the data models, reconciliating and aggregating data manually, creating applications exploiting low level Web Service and/or REST Call without the support of development tools, and neither of a semantic data aggregator. In this paper, an innovative tool for smart city web and mobile Apps development is presented. It exploits the Km4City data aggregator and semantic model, and includes: (i) tools for assisting developers for generating calls to Smart City API via visual queries on a graphical user interface; and (ii) open source Apps Development Kit for shortening the development. Finally, the paper reports about experimental results performed in the usage of the tool in the context of a large national project called Sii-Mobility which involved several developers. The same approach is presently adopted by other large projects as well.

Keywords — smart city, smart city development tools, smart city API

I. INTRODUCTION

Most of the smart city solutions must cope with big data aspects as data volume, variety, and veracity [4]. Open data, as static data, are not typically the main source of information in the city in terms of volume, neither the most valuable for the city users (citizens, tourists, commuters, operators, students, etc.). Most of the big data problems and values in the smart city platforms are related to real time data as the public transports, vehicle and human mobility in city, events, parking, weather, wind, first aid triage, etc. A smart city architecture should be capable to take advantage of huge amount of big data coming from several domains, at different velocity for exploiting and analysing them for computing integrated and multidomain information, making predictions, detecting anomalies for early warning and for producing suggestions and recommendations to city users and operators.

In the last years, many architectural solutions have been proposed with the aim of making data accessible, aggregated, usable, and exploitable, etc. [1], [2], [3], [5], and many of them failed in posing the basis for creating a smart city open environment for new and smart applications. It is obvious to state that, cloud and distributed systems approaches are at the basis of the big data solutions provided for smart city. On the other hand, the city infrastructure is much more complex, and the limited focus on only some of the above-mentioned aspects would create limitations not accepted for the city operators and for the city development of smart services.

In most cases, the effectiveness of data service system for Smart City is enabled by the availability of private data owned and managed by City Operators addressing specific domains: mobility operator, energy providers, business services (health, water), telecom operators, tourist operators, universities, etc. In most cases, the data are collected into open data portals such as CKAN [http://ckan.org], OpenDataSoft [6], ArcGIS and OpenData [7], SOCRATA [8] also based on ArcGIS. On the other hand, most of the open data portals are unsuitable for exposing semantic interoperable API (application program interface) to facilitate the production of web and mobile Apps, and for managing real time data as those produced by IoT solutions. IoT solutions are typically focussed on producing and collecting data according to Push/Pull approaches on data brokers (see ActiveMQ Apache, ORION FiWare, Kafka Apache, etc.) using several different kinds of protocols and network levels. At the end, the above mentioned Open Data portal models and data brokers may provide some API for direct access to data tables without a semantic integration expositing reconciliated data for geolocation, identifier, relationships, time, etc. [9]. With the aim of passing from data to services, a number of Smart City API solutions have been proposed on different domains of the smart city such as: CitySDK [10], EPIC [11], Transport.API [12], Navitia.io [6], and Km4City [9]. Moreover, Smart City API systems are in general a very complex to be exploited, especially if they refer to a great amount of different and complex aggregated data as those managed by multidomain solutions, see for example EO15 [13], well-shaped but complex ecosystem. In this sense, also other smart city API solutions are still a combination of different APIs.

Despite of the above described large offer of different smart city architectures and solutions, nothing or very few has been done on making simple the creation of mobile and web applications. The smart city developers, typically SME, researchers, students, and operators, still have to develop their applications by studying in deep the data sets and provided API for recovering data models, and reconciliating and aggregating data (to be repeated at each changing of the data model, and for each dataset), creating applications exploiting low level Web Service and/or REST Call without the support of comprehensive development tools for Apps [15], [14], [1], [http://ckan.org]. In addition, if the data are coming in real time as streams, or may change several times per day (sometimes at model and protocol levels). Thus, a real time data aggregation

server or streaming processes is needed also addressing the real time reconciliation of data. On the other hand, the world of mobile and web Apps is changing, the Apps are becoming more and more dynamic, pushing on HTML5 and on instant Apps, as the Android Instant Apps aims at running them without installation. This approach will create the needs to a continuous renovation of Apps and the reduction of a stable set of users. In conclusion, the production of web and mobile Apps has to be faster and cost effective, and probably smaller and cross linked among each other.

In this paper, an innovative tool for developing smart city web and mobile Apps is presented. It includes: (i) a set of tools for assisting developers in understanding the knowledge model and for generating Smart City API calls by performing visual query on a graphical user interface; and (ii) a set of open source Apps for shortening the development (e.g., starting from scratch on new kind of Apps, as well as developing modules that can be loaded dynamically from an already installed App) [16]. The proposed development tool is based on Smart City API described in [9] which in turn are based on Km4City (Knowledge Model 4 the City) ontology [17] and RDF storage. The proposed development tools have been realized in the context of Sii-Mobility Smart City national project on mobility and transport of Italian Ministry of Industry and Research, and presently also used as development tool and model in REPLICATE H2020, and RESOLUTE H2020 projects of the European Commission.

The paper is organized as follows. In Section II, the Sii-Mobility architecture is described. Section III presents the smart city development system for supporting the developers in understanding the data and models, and generating REST Calls to Smart City API. In Section IV, the architecture of the modular Smart City App model (Application Development Kit, ADK) is presented to allow the dynamic loading of modules, enabling user behaviour tracking and analysis, monitoring user appreciation of features, etc. Section V presents some experimental results obtained during validation with the user group of developers, and by monitoring the real usage of the Apps with real city users. Conclusions are drawn in Section VI.

II. SII-MOBILITY ARCHITECTURE

The reference architecture of Sii-Mobility smart city solution is depicted in Figure 1. The solution allows to collect data coming from different kind of sources (open data, private data, real time data), belonging to different domains (mobility, environment, energy, culture, e-health, weather, etc.), and by means of different protocols of different levels (several IOT, DATEX [18], Rest, WS, ETSI, OpenM2M, etc.). The architecture is based on a semantic aggregation of data and services according to the Km4City ontological model. Data providers as City Operators and Data Brokers offer data are collected by the smart city: (i) in pull by using Extract Transform and Load (ETL), (ii) in push/streaming via dedicated stream lines and processes. Among the data collected those provided in open data from the: local municipalities, Tuscany region (Observatory of mobility, MIIC), LAMMA weather agency, ARPAT environmental agency, etc., and several private data coming from City Operators: mobility, energy, health, cultural heritage, services, tourism, wine and food services, education, wellness, etc. Moreover, Data Brokers collect and manage real time data coming from sensors (IoT), and from vehicular kits (On board Device) which are developed for monitoring and informing car, bus and bike drivers, etc. Once the data are collected, the back office executes several processes for: improving data quality, reconciliating data, and converting data into triples for the RDF store of the Knowledge Base, KB [19], implemented by using a Virtuoso triple store.

All the above mentioned processes are scheduled on the Big Data processing back office based on a Distributed Smart City Engine Scheduler (DISCES) tool developed for Sii-Mobility and made open source. DISCES uses several virtual machines allocated on the cloud according to their schedule and requests arriving from the Decision Makers, Developers and Data Analytics (typically 3.5-5 thousand of jobs per day). The data collection processes are scheduled according to different policies, for Open Data (the solution re-upload when they change), quasi real time data (change a few times per day), real time data (change every few seconds, such as the position of the Bus, or the position of the City Users) and taking into account all the permissions access connected to each different piece of information managed in the Km4City Knowledge base.

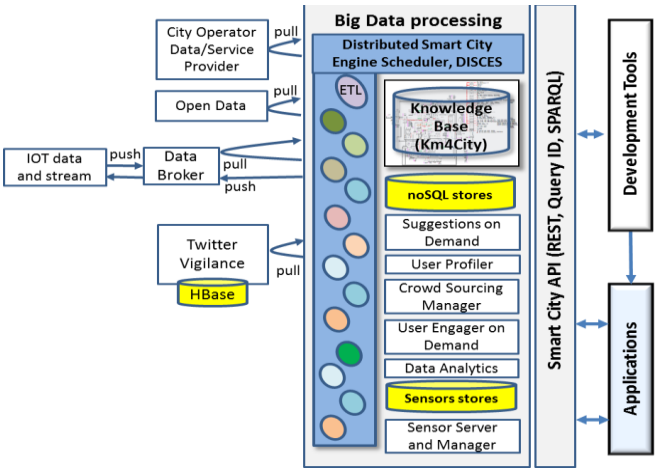

Figure 1- Sii-Mobility Architecture

For semantic aggregation of data and services, the Km4City Ontology (http://www.km4city.org) [17], [19] has been adopted and extended by adding details regarding mobility and transport, sensors, environment, with respect to former model. Now Km4City is modeling multiple domain aspects related to mobility, services, Wi-Fi, cultural services, energy, structure (streets, civic numbers, green areas, sensors, busses, etc.).

Among the complex data that need to be collected are those coming from social media. In this case, from Twitter via Twitter Vigilance platform regarding the monitoring of some of the topics/"channels" and users related to city services on mobility and traffic in the city, and on alerts, civil protection messages, weather forecast (http://www.disit.org/tv).

The above data, collected and aggregated, is exploited by a number of scheduled data analytics processes to compute contextualized: user behaviour and mobility analysis, user recommendations, suggestions and personal assistant messages according to the city and city operator strategies.

Therefore, in order to be capable of providing contextualized information web and mobile Apps collect a

number of data provide them to Sii-Mobility via the Sensor Server and Manager, SSM. The data collected from Apps (mainly mobiles) are related to many different aspects: the position of the city users, preferences (user profiles), requests to the Smart City API, searching queries, action performed on mobile, velocity, accelerations, etc. [20]. This kinds of data allows to understand the user behaviour, and thus, to engage the users generating ad-hoc and contextualized suggestions and recommendations. For example, detecting when a city user has less sustainable behaviour for its life and for the city.

Sii-Mobility architecture, in addition to the RDF store for the knowledge base (developed on top of Virtuoso), presents several noSQL stores (namely: HBase and Mongo) for storing tabular data as those arriving from SSM, IOT, streams, and to make versioning of collected data that have to be passed into the RDF store for reasoning. This approach allows accessing tabular data for Data Analytics processes such as those performed for the: estimations of recommendations, engagements, traffic flow predictions, parking forecast, clustering of sensor data behaviour, and anomaly detection. When needed, federated queries can be performed among RDF and tabular stores. The resulted architecture provided several services via Smart City API to Development Tools or to the city users tools (Applications). The development tools are described in the next Section, and allow visually formalizing queries to generate Smart City API REST calls. The generated calls are sent to the developers via email for shortening the production of web and mobile Apps as described in the sequel. The ServiceMap development tool has been used to create multiplatform Apps, such as "Florence what where", available on all the mobile platforms, on HTML5 and Windows 10 (http://www.disit.org/app). The mobile App and of the Smart City Control Room Dashboard (not described in this paper) are the main sources of workload for the smart city APIs.

III. Smart City App Development system

The main purpose of the Sii-Mobility architecture is to enrich and aggregate the data, thanks to the Km4City semantic Model, and then make the data available for other purposes, depending on the permission access of each different kind of data. Thanks to the visual approach the understanding and access to geo-referenced information stored in the Km4City KB results to be simplified by a set of services and tools. These tools have been developed on top of the Smart City API to simplify the work of developers, with an easy visual connection on data. The main developers' tools created to solve these kinds of problems are: i) ServiceMap [22]; ii) Application Developer Kit, ADK [16]. The set of development tools also includes tools to work with RDF stores (also visually generating query examples in SPARQL) and a direct interface for formulating SPARQL queries, when needed.

In **Figure 2**, the interaction among the main tools is depicted. The actions performed by the developers to create their application are: i) search on the ServiceMap and visualize the resulting data on the map; ii) save the searches done and receive an e-mail including the API calls to be used in the App to get the same data obtained by the ServiceMap; iii) use the API calls to develop web or mobile Apps, as well as to use the ADK to create new dynamic modules.

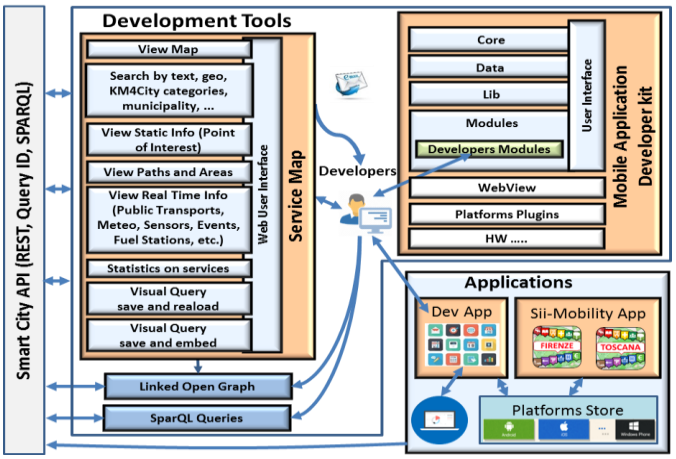

Figure 2 – ServiceMap Development tools

The ServiceMap [22], [25] architecture is grounded on the Km4City KB than can be questioned through the Smart City API or directly making SPARQL queries on it (see Figures 1 and 2). When the developers pose queries on the ServiceMap it sends requests which are translated into SPARQL queries or Smart City API calls (also converted in SPARQL and/or federated queries). Resulting queries are posed to KB and NoSQL stores. Then the results are processed to convert them in HTML and/or JSON returning them to the ServiceMap user. The ServiceMap has been developed by using JSP application. In order to create interactive and enriched maps JavaScript library Leaflet [21] has been adopted. Thanks to the well-documented API and numerous plugins, Leaflet can build a map usable and rich in detail, making use of maps released by OpenStreetMap.

A. ServiceMap tool for Visual Generation of Calls

The ServiceMap allows developers to:

1. **Search**: Make a set of different searches on the data collected in the Km4City KB visually (each info is geo-localized and can be viewed in the map);
2. **Save&standardAPI**: Save the performed visual query to receive via e-mail the Rest calls to the APIs to obtain the same data [20], according to the Smart City API syntax with all parameters explicated. The calls can be directly used into web and mobile Apps, and can be used also to learn how to query the Smart City API;
3. **Save&QueryID_API**: Save the performed visual query to receive via e-mail a Rest call with a simplified syntax referring to the so called QueryID_API only. This id refers to a full Rest Call saved on the server. The simplified syntax of Rest call with QueryID_API can be used to avoid the usage of complex syntax and to allow changing the query on server without redeploying the application;
4. **Save&EMBED**: Save the experience/queries of the users in visually recall smart city elements on the map, and thus to give the possibility of embedding the view on a third party web page;
5. **Search&LOG**: Access to the Linked Open Graph tool (LOG) [24], to open a visual description of the KB elements and their relationships, which is the Km4City Semantic Model in a graphical modality.

Moreover, the developers can also use a tool for to directly pose SPARQL Queries on the Km4City KB, and verifying the licensing level [23] http://log.disit.org/sparql_query_frontend/.

Developers' Action 1: Search

ServiceMap allows the developers to make geographical queries on the Km4City KB and provide them results on a map, on the basis of: Street Graph from Tuscany region, Open Data from Florence Municipality, traffic monitoring, geo and weather forecast information from LAMMA, traffic sensors, services, events, car park occupancy, timetable/routes/stops of Public Transport Lines, gas stations in Italy, pollution information (Air quality) from ARPAT, first aid status of major hospitals, etc. Some of data are provided in a real time modality, such as the occupancy of the car parks, the traffic sensors, the events, the weather forecast in Tuscany, the first air, pollution, etc.

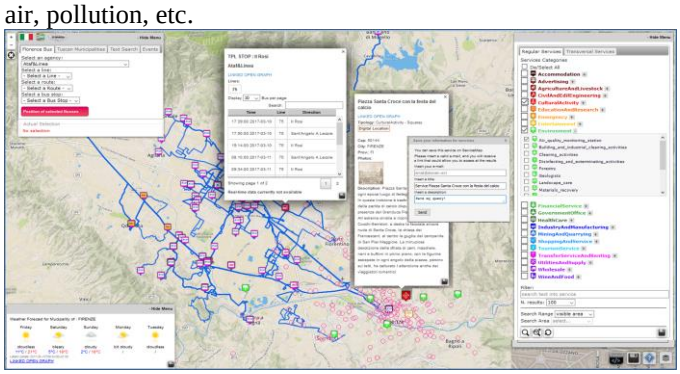

Figure 3 - ServiceMap Web Interface.

The ServiceMap interface provides a number of search capabilities such as: i) search geo-located elements (also called Services) for municipality; ii) search Services for proximity to a point on the map or around a reference Service; iii) search Public Transport Routes and stops; iv) Search Public Transport Line for proximity to a point on the map or to a Service; v) Search Events in the period; vi) Search Services and Events by Text, vii) Save a search made and reloading a saved query by query-ID, see **Figure 3.** These kinds of features are offered via two main menus. The menus and thus the taxonomy of Service of Km4City are available in both English and Italian language.

Therefore, the Top Left Menu contains 4 tabs, each of them provides a specific modality to make a search:

- Transport Public Lines (TPL): to select a TPL agency, line and route, and view on map (both routes and stops).
- Tuscan Municipalities: to select local govern areas in which the Services are searched. Once selected a local govern, the corresponding weather forecast is automatically shown in the lower left corner. In addition, the user may perform a filtering on the basis of the Service categories listed on the Top Right menu.
- Text search: to perform a search on the whole KB, by using full text.
- Events: to search the events that take place in the area, specifying the time interval of interest: day, week or month.

In all cases, the query results appear on the map as a list of events with their markers. And in all cases the user can: (i) restrict the search to a delimited area choosing the center with latitude and longitude as decimal degrees and the radius in Km, (ii) limit the maximum number of results. For case (i) a reference point in the map has to be chosen; thus, the ServiceMap informs the user about the selected GPS coordinates, the closest street addresses, and the closest other structures located in the neighbouring. In most of the above cases, the interface proposes a small disk icon in the dialog boxes. This icon allows the user to request at the ServiceMap to send an email with the examples in both JSON and HTML.
In the same context, when selected a point or a Service in the map, it is also possible to select a second point or Service and request for producing a call to Smart City API of routing. Presently, this service is provided only for pedestrian navigation in Florence downtown, providing the shortest path.

The Top Right Menu in ServiceMap provides two main Tabs:

1) Regular Services: This menu is composed of 20 checkboxes corresponding to the taxonomy Services as described by Km4City ontology. It includes macro-categories (e.g.: Cultural Activity, Tourism Service, Environment, etc.) and a number of categories for each of them (e.g.: Museum, Restaurant, Hotel, Pharmacy etc.). At the moment, there are more than 530 categories dynamically loaded. The search for proximity to a point on the map allows to choose the maximum number of results desired in output (between the values 10, 20, 50, 100, 200, 500 and No Limit) and the search radius (between the values 100m, 200m, 500m, 1km, 2km, 5km, all the Visible Area in the Map, on Specific areas or along a line). Searches on specific area/line allow performing a query on recalling all the selected services located in formalized shape (closed or lines). This feature exploits the Smart City API which allows searching for specific Services: (i) along a line (e.g., cycling path, a tram line, route to home); (ii) into a specific are providing a shape data.
The additional functionalities on the same panel allows: to clean the view, to ask for the histogram distribution of Services in the selected area, and to ask to the email with the Smart City APP Rest call exemplified.

2) Transversal Services: The modality of search is similar of those proposed for Regular Services. In this case, the most relevant categories are:

- Areas: which are Services endowed of a closed shape (singly or multiple-connected), as Gardens, Controlled Parking Zone, Green and Sports Areas whose shape be highlighted on the map.
- Digital Location: which includes different type of services that can contain various multimedia content such as jpeg images, audio files or pdf attachments, multilingual descriptions, etc.
- Happening now (Events): daily, weekly, monthly, etc.
- Path and Public Transport Lines, which are Services presenting a path, such as: Tourist Trails, Cycle Paths or Bus/Train/Tram/Ferry Routes.
- Sensors which are IOT Services such as: traffic flow sensors, environmental sensors, bus-stops, smart benches, smart lights, parking, etc.

- First Aid: main hospitals providing in real time the status of occupancy of the emergency rooms triage.
- Bus/Tram/Train/Ferry stops: which are paths and bus stops of each Service passing in a given area around a point. This gives you an example of the API to search for a bus to reach a given point in the city.

Developers' Action 2/3: Save&standardAPI and Save&QueryID_API:

Once the developers have searched and visualized data in the ServiceMap, they can save their queries on server by clicking on the save buttons (an icon disk located in each menu). After this action a popup dialog appears, asking for (i) email address, (ii) a reference title; and (iii) a description of which service/s the user wants to save (e.g., Piazza Santa Croce, in Figure 3). Thus the query is saved on server for a further reuse and a set of examples are sent via email to the developer. The examples received via email describe specific API REST calls that can be used to invoke the Smart City API from Apps, to obtain the data interest. The Smart City API Rest call provided are in the format of:

- **Standard API:** read only links (providing information in an HTML and/or JSON);
- **Query_ID API**: **read** and **write** links that are saved on server side and can be used in the Applications. This approach has two advantages for the developer which: (i) avoids the usage of complex standard API in the code, (ii) may change query associated with Query_ID on the server by using the ServiceMap without the need of rebuilding and redistributing the Web or Mobile App.

According to the latter case, the developer receives from the two different Rest call: (read) to use the Query_ID API call in the App; (write) to automatically open the Servicemap and change the associated query. It is relevant to remark that in all cases the data requested are dynamically updated at the moment of the call, showing to the final users the updated information.

Developers' Action 5: Search&LOG:

In order to allow developers to learn/create formalizing SPARQL queries for a particular problem that cannot be solved using the provided Smart City APIs the developer needs to understand the Km4City ontology and data available in the RDF store. To this end, the ServiceMap allows exploring the KB in the context of each Service identified on the map by using the Linked Open Graph (LOG) tool [24]. Starting from a search result (e.g., a single service) it is possible to explore the KB starting from this point allowing to see the relations with other entities in the KB, eventually hidden from the user interface for simplification. For example, it is possible to understand how the Services are connected with the road graph elements or how to access more complete real time information about IOT sensors. The LOG allows to visually navigating all the relations among the entities as depicted in **Figure 4**, including those bringing to other LOD/LD (linked open data, linked data) resources as dbPedia.

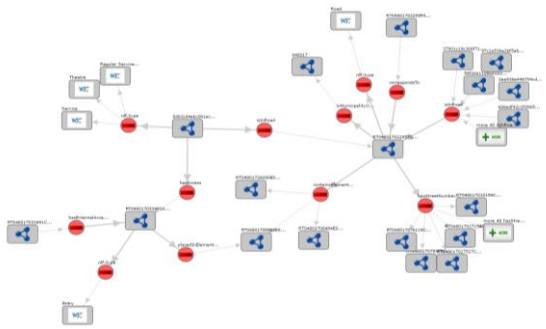

Figure 4 – Linked Open Graph

IV. Application Development Kit, ADK

The main aim of the Application Developer Kit, ADK, consists in making simple and fast the production of new modules in complex Apps. This is possible thanks to the ServiceMap development tool and to the proposed architecture for the web and mobile Apps. The main requirements that have drown the design of ADK have been the strong need of:

Modularity: to have a structure which allows the modularization of functionalities in order to make the code development for modules simpler, and allocated to several distinct teams as one may have on distributed projects with multipartner and/or in a smart city development with multiple operators.

Dynamicity: to make possible the addition/update/ removal of modules to the Apps at run time (when the Apps are installed in the device of the final user) in order to (i) speed up the deploy of the new functionality in the hands of the final users (among them also special Modules for managing events, or critical cases), (ii) maintain the attention of the App users with special temporary games, incentives, trials, etc., and in App commercial functionalities, if needed.

Personalization: to adapt the user experience (menu, user interface, functionalities) on the basis of the users' actions and profile; also giving at the user the possibility of resetting and changing the menu setting, functionality preferences, etc. See for example, the customization obtained with Telegram Bot.

Alerting: to enable the App to receive alerts, notifications and messages possibly when the App is in background and thus off. So that to inform the user about critical situations in the city and also for personal assistance, suggestions, etc. This is feature that may have different possible implementation on different platforms.

Multiplatform: to realize Apps that can be installed and performed on more than one platform, providing a similar level of experience, and avoiding substantial changes of the code to maintain limited the development costs.

The proposed architecture of the Application Developer Kit, ADK, and thus of the final App have been designed to enable every developer to create his own module, that can be dynamically loaded inside the application.

Modularity and Dynamicity

The **modular structure of the ADK** (see Figure 5) is used to prevent the change of files inside the application with another code or data, from new developers. New modules have to be placed inside the modules folder of the ADK, and the ADK

has to be loaded at the proper time scripts and templates. New data have to be also combined with data already present in the basic version of the ADK (i.e., Labels and Alerts in Data block). The modularity structure does not allow developers to modify other parts of the ADK. Developers can use the functionalities made available for the modules and submitted from other scripts (i.e., take data from GPS position, visualize services on the map and other functionalities offered by scripts contained in Core block). As shown in Figure 5, the block Modules contains those developed and integrated on Sii-Mobility application concerning functionalities that offer data in real time:

- Parking Searcher: to find closest and freest car parks in the city, showing distance and number of free parks.
- Fuel Station Searcher: to find closest and cheapest Fuel Stations showing the favourite fuel kind and price.
- TPL Searcher: to find the closer public transportations stops and showing the next ride time, or the full list;
- First Aid Searcher: to find hospitals and showing the emergency room (first aid) situation, the triage status;
- Pollution Searcher: to find the closest environmental and pollution sensors and shows the last data values collected by the sensor(s);
- Others such as: weather forecast, events in the city, civil protection information alert, etc.

Each module is substantially a mini-application in HTML5. In a certain sense, each of them exploits the Smart City API, the registration of the user ID, the access to sensors of the device, etc., and thus may receive information from the Sii-Mobility servers (see Figure 1). The produced modules may be dynamically loaded inside the official application or provided into the App from the store. In the first case, every time the App is executed, it looks for the availability of new modules to load them from the server site, also updating the former versions when needed. The module loader directly locates the new and updated function and mini-apps on the menu, and when needed new buttons are connected to the new feature that will appear and in the desired place according to the server manifest of the module.

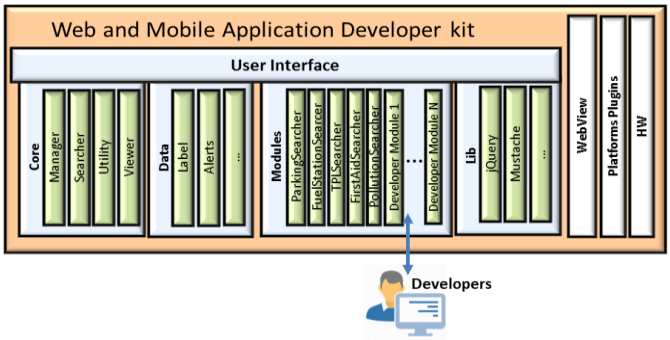

Figure 5 – Web and Mobile Application developer kit.

Personalization and Profiling

Apps Users are profiled and their profile is communicated to the server. Users' profiles are presently kept totally anonymous and classified as citizens, students, commuters or tourists, according to user preferences. Thus, their interaction with the interface is anonymously recorded, to know the most sought services of the different profiles and modify the main menu. This approach allows us to tune the service according to the collective and personal user behaviour, with the aim of improving the user experience providing personalized (i) main menu arrangement of the functionalities according to the user profile (the user also may change it to have precise personalization), (ii) suggestions (the user may decide to ban some of the suggestion or even a category), (iii) menu search among categories (the user may decide to make further selection). Therefore, the menus offered for each profile kind is modified according to the statistics calculated on the collected data, by defining the positions the most researched categories, buttons, etc., and in some cases, putting off/on some functionality. For example, the triage monitoring may be of interest for citizens and operators and less for tourists, the parking status is interesting for users using the private car and less for those moving with the public busses or in bike, etc.

The priority on the creation of the menu profiles is given in first place to the updated menu which can be found on the server. If the server is unreachable, the last saved menu for that specific profile is loaded; if the menu has not been saved, the loaded menu is the menu released with the application update, and so on.

Alerting and tracking

The city users need to be informed on what is going on the city. On the other hand, most of the mobile platforms have different approaches for providing asynchronous notifications in push to the device when the App is not in foreground, is not executed by the user. On this regard, Android allows executing processes in background so that it can be used to collect changes with a polling approach. On the other hand, iOS provides a service called APN for central management of notifications in push and does not allow executing processes in background. A similar approach is also provided for Windows Phone. In addition to the alerting, the movements of the device should be also communicated to the server in order to get new context based suggestions and alerting, such as: please take care about the weather forecast in your area, alarms of civil protection, environmental status, closer car park, etc. Some of these innovative and smart features are produced as suggestion, engagements from personal assistant, predictions produced by the data analytics modules (e.g., parking, arrival of busses, etc.), alerts (e.g., civil protections, changes in the traffic, events). A new and additional feature of notification can be obtained integrating Telegram, which has to be installed into the App. In this case, the server identifying the critical situation has to interact with Telegram server. This allow to send notifications to Apple and WinPhone platforms.

Multiplatform

In order to satisfy the first requirement of multiplatform, the ADK and the architecture have been based on Apache Cordova framework. It allows realizing hybrid applications on multiple platforms. The applications developed with Apache Cordova, shell consists of user-interface implemented with HTML, CSS and JavaScript, and of plugins which allow to use the specific hardware functionalities from the different platforms (i.e., battery-status, camera, device orientation) (Platforms Plugins in Figure 5), through a JavaScript interface.

The Sii-Mobility App development has shown the powerful of this framework that permit in a very short time (i.e., one day or max one week) to develop and publish the application on different platforms stores. Therefore, it is possible to have almost the same code for all platforms. It is possible to start from the ADK source code released as Affero GPL, available on https://github.com/disit/siiMobilityAppKit, for all developers that would like to realize new modules for Sii-Mobility official applications on the basis of data available through Smart City API Km4City and/or from other sources and APIs that may be related to private data of the city operators.

A. Openness of the Smart City API

ServiceMap and the Smart City API may be also invoked by other kinds of Apps and not only by those adopting the approach of developing modules for the main Apps. In this case, the ADK can be used for taking inspiration only since the license cannot accept to extract modules and create new Apps.

V. Experimental Results

The ServiceMap is an experimental platform since the 2015 providing always a growing number of functionalities and data kinds. In this phase of the experimentation, a number of developers are using the development platform of Sii-Mobility. For the validation, in January 2017 an internal hackathon with 8 groups (1-2 developer) of industrial partner has been organized. They have been engaged on developing the same module (namely the gasoline prices) producing it in a quite similar shape as that published on the store, in about 3.5 hours of work, after attended a short training section of 2 hours (see http://www.sii-mobility.org/index.php/eventi/mobile-app-workshop). This demonstrated the effectiveness of the proposed approach. A new round of hackathon is planned for April 2017 (see Http://www.sii-mobility.org), open to external partners and thus for modules and completely new apps inspired by the approach.

In Table I, the number of requests performed on ServiceMap from Developers (excluding our developers and users) is reported. Users are using the GUI and APIs to access to the different features available for accessing the data available on the KB. In particular, developers used the ServiceMap GUI functionalities 6,375 times in the period from Dec. 1st 2016 - Mar. 9th 2017 to perform queries and to study the data available on the system. These developers have created in this period of time 133 different *queryIds* that have been used via API 1,633 times. However, the most used API type is the REST API that provides a JSON data structure, these kinds of APIs were used 676,204 times (95.95%). The most advanced SPARQL API allowing querying directly to the Km4City KB has been used only 19,296 times (2.74%). Requests at the Smart City API for getting results in HTML, typically used to Embed ServiceMap in other web pages, with Km4City data, have been used in the same period of time for a total of 1,202 times (0.17%).

TABLE I. Number of requests on Smart City API and development tools performed between Dec 1st 2016 and Mar 9th 2017

Request type	#requests	%
API (no queryId)	676,204	95.95%
SPARQL call	19,296	2.74%
ServiceMap GUI	6,375	0.9%
API via queryId	1,633	0.23%
API HTML	1,202	0.17%

According to Table I, the classical Rest API style is the mostly used. From a deeper analysis of the functionalities used in the ServiceMap GUI it comes out that the most used function is the information on a single Service (point of interest) with 38%, and the second one is the location information on a GPS position with 27% (returning the address and geometries hitting the GPS position). The third mostly used functionality is the search of Services by GPS position with a 19%, while at the 8% the search by municipality, and at 5% the text search. Other functionalities are below the 1%. Regarding the saved queries, following the Query ID approach of API and App, the 24% are queries to search Services via GPS/service position while 10% to search Services on a municipality and 35% are for looking for a specific Service, while the 26% are to save a complex configuration to be embedded in an html page. For SPARQL API requests, 19,296 requests are coming from 82 different IP addresses meaning that these queries are typically performed on the server side (mainly from other servers and tools and not from mobiles also detected from the User Agent analysis). The Mobile Apps used the REST APIs in the 130,820 times over the 676,204 requests covering the 19% of the requests from 1729 different users.

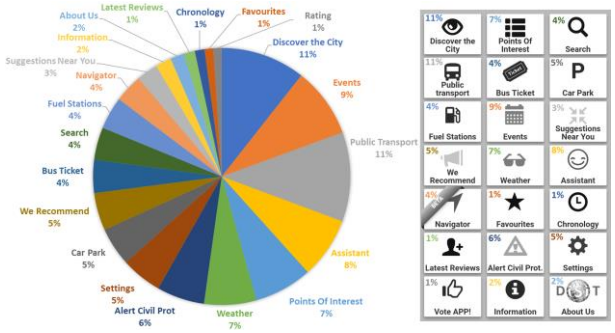

Figure 6 – Statistics on clicks of the main menu

In **Figure 6**, a statistic on users' clicks on the buttons of the main menu is presented. In the figure, the percentages of clicks have been depicted in the buttons to see the spatial distribution. Most of the selections are concentrated in the button located on top left because the users search the mainly functionality on this area (this is also due to the arranged performed on the basis of an early analysis performed in mid of 2016). Secondly, the clicks are focused on the central buttons and on the alert button: on these buttons during the use of the application are added badges/labels highlighting the number of news the user may found accessing to that feature (i.e., the number of events planned for that day, the number of messages sent by the assistant, number of suggestions, etc.). Then the notifications appearing on buttons provokes curiosity about users who are inclined to click to see what shows them.

In **Figure 7**, a statistic on the categories requested by users is depicted. The most searched categories are those relating to buttons that display directly to user the services sought (buttons on the principal menu or buttons aside of the map,

when it is shown). Very popular is also the function “Around You” that it is shown to the user in a popup over the GPS markers and over manual position marker. Search for categories, represented in the graph by the labels "Tourist Menu" and "Citizen Menu" is not much used even though allows the user a more targeted choice of services that should be searched.

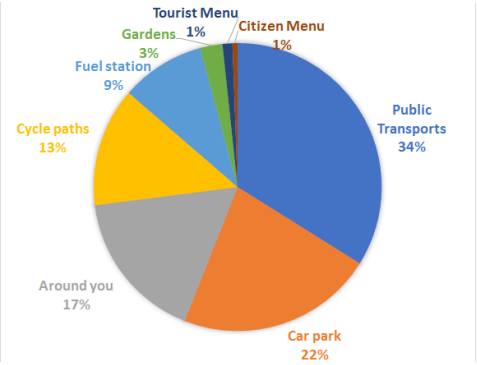

Figure 7 - Statistics on more researched categories

VI. CONCLUSIONS

The development of smart city App is a time consuming activity and is becoming every day more dynamic and frequent. Data and API are in most cases offered from classic Open Data portals; while they are unsuitable for developing smart Apps (lack of semantic aggregation and interoperability of data, lack of data analytics support). Thus developers as: SME, researchers, students, and operators, had to develop Apps by studying data and complex APIs before becoming productive. To cope with these problems, an innovative development approach for smart city web and mobile Apps has been designed and experimented in a real context. It exploits the Km4City data aggregator and semantic model, and includes: (i) tools for assisting developers for generating Smart City API calls via visual query on a graphical user interface; and (ii) open source Apps for shortening the development of mobile App modules. The paper reported data regarding the effectiveness of the time saved in the development and the assessment of the user actions on the mobile App and development tools. The results have been assessed on the context of a large national project called Sii-Mobility which involved several developers. The same approach is presently adopted, used, and contributed by other large projects as well, namely: RESOLUTE 2020 and REPLICATE H2020. They contributed with new data and detailed features, and some new functionality to the general solution.

ACKNOWLEDGMENT

The authors would like to thanks the MIUR, to the University of Florence and companies involved for co-founding of Sii-Mobility Project. Km4City is an open technology of research created by DISIT Lab.

REFERENCES

[1] Anthopoulos, Leonidas, and Panos Fitsilis. "Exploring architectural and organizational features in smart cities." Advanced Communication Technology (ICACT), 2014 16th Int. Conference on. IEEE, 2014.

[2] Filipponi, L.; Vitaletti, A.; Landi, G.; Memeo, V.; Laura, G.; Pucci, P., "Smart City: An Event Driven Architecture for Monitoring Public Spaces with Heterogeneous Sensors," in Sensor Technologies and Applications (SENSORCOMM), 2010, vol., no., pp.281-286, 18-25 July 2010.

[3] Domingo, A.; Bellalta, B.; Palacin, M.; Oliver, M.; Almirall, E., "Public Open Sensor Data: Revolutionizing Smart Cities," in Tech. and Society Magazine, IEEE , vol.32, no.4, pp.50-56, 2013.

[4] P. Bellini, M. Di Claudio, P. Nesi, N. Rauch, "Taxonomy and Review of Big Data Solutions Navigation", as Chapter 2 in "Big Data Computing", Ed. Rajendra Akerkar, Western Norway Research Institute, Norway, Chapman and Hall/CRC press, ISBN 978-1-46-657837-1

[5] Chourabi, Hafedh, et al. "Understanding smart cities: An integrative framework." System Science (HICSS), 2012 45th Hawaii International Conference on. IEEE, 2012.

[6] OpenDataSoft: https://www.opendatasoft.com/

[7] ArcGIS OpenData: http://opendata.arcgis.com/

[8] SOCRATA: https://www.socrata.com/

[9] C. Badii, P. Bellini, D. Cenni, G. Martelli, P. Nesi, M. Paolucci, "Km4City Smart City API: an integrated support for mobility services", 2nd IEEE Int. Conf. on Smart Computing (SMARTCOMP 2016), St. Louis, USA, 18-20 May 2016.

[10] CitySDK: http://www.citysdk.eu

[11] EPIC, European Platform for Intelligent Cities, http://www.epic-cities.eu, ICT PSP (2011-2013)

[12] Transport.API, http://www.transportapi.com

[13] E015 digital ecosystem, http://www.e015.expo2015.org/

[14] IBM Institute for Business Value, “How Smart is your city? Helping cities measure progress”, [online]. http://www.ibm.com/smarterplanet/global/files/uk__en_uk__cities__ibm_sp_pov_smartcity.pdf

[15] Alcatel-Lucent Market and Consumer Insight team, “Getting Smart about Smart Cities Understanding the market opportunity in the cities of tomorrow”, Oct. 2013

[16] Sii-Mobility APP Kit: APP module development http://www.disit.org/6992

[17] P. Bellini, M. Benigni, R. Billero, P. Nesi and N. Rauch, "Km4City Ontology Building vs Data Harvesting and Cleaning for Smart-city Services", International Journal of Visual Language and Computing, Elsevier, 2014.

[18] DATEX II: http://www.datex2.eu/sites/www.datex2.eu/files/Datex_Brochure_2011.pdf

[19] P. Bellini, I. Bruno, P. Nesi, N. Rauch, "Graph Databases Methodology and Tool Supporting Index/Store Versioning", publication on JVLC, Journal of Visual Languages and Computing, Elsevier, 2015.

[20] Sii-Mobility Km4City: Smart City API Documentation http://www.disit.org/6991

[21] “Leaflet API reference,”: http://leafletjs.com/reference.html .

[22] Servicemap development tool, http://servicemap.disit.org

[23] P. Bellini, L. Bertocci, F. Betti, P. Nesi, "Rights Enforcement and Licensing Understanding for RDF Stores Aggregating Open and Private Data Sets", 2nd IEEE Int. Smart Cities Conference (ISC2 2016), September 2016, Trento, Italy

[24] Linked Open Graph, http://log.disit.org/service/

[25] C. Badii, P. Bellini, D. Cenni, A. Difino, P. Nesi, M. Paolucci, Analysis and Assessment of a Knowledge Based Smart City Architecture Providing Service APIs, Future Generation Computer Systems, Elsevier.