Model-Driven Mobile CrowdSensing for Smart Cities

Paulo César F. Melo, Fábio M. Costa

Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74.690-900 – Goiânia – GO – Brazil

{paulomelo,fmc}@inf.ufg.br

Abstract. *Making cities smarter can help improve city services, optimize resource and infrastructure utilization and increase quality of life. Smart Cities connect citizens in novel ways by leveraging the latest advances in information and communication technologies (ICT). The integration of rich sensing capabilities in today's mobile devices allows their users to actively participate in sensing the environment. In Mobile CrowdSensing (MCS) citizens of a Smart City collect, share and jointly use services based on sensed data. The main challenges for smart cities regarding MCS is the heterogeneity of devices and the dynamism of the environment. To overcome these challenges, this paper presents an architecture based on models at runtime (M@rt) to support dynamic MCS queries in Smart Cities. The architecture is proposed as an extension of the InterSCity platform, leveraging on its existing services and on its capability to integrate city infrastructure resources.*

1. Introduction

Modern-time cities face challenges to achieve goals related to socio-economic development and quality of life, notably due to the concentration of the population and the pressures that arise from it. The concept of Smart City was proposed in response to these challenges (Celino et al., 2013). One of its main themes is the integration of the physical and virtual worlds (Borgia et al., 2014). This integration is achieved with the introduction of capabilities for environmental sensing and actuation, allowing capture, analysis and processing of real-world data, transforming the data into useful information and allowing autonomic interventions in the urban space. Thus, smart city resources, also called things in an Internet of Things perspective, are equipped with sensing and/or actuation capabilities, along with communication capabilities to share information.

In this context, smart cities need to take advantage not only from information collected from sensors that belong to its infrastructure, but also from the mobile devices owned by its citizens, which increasingly have advances sensing capabilities (e.g. cameras, microphone, accelerometer, GPS). In Mobile CrowdSensing (MCS) citizens of a smart city collect, share and jointly use services based on community-sensed data (Stojanovic et al., 2016).

Smart cities have a wide range of domains, such as MCS, and these domains can be integrated into a complete and consistent solution as part of a software platform, which includes foundation services for the development, integration, deployment, and management smart city applications. In the MCS domain, the development of smart city platforms to support applications poses a number of challenges, such as interoperability among different resources, recruiting of appropriate data sources, collection and processing of data from those sources, and runtime adaptation of the applications in dynamic environments (Alvear et al., 2018).

To help overcome these challenges, this paper proposes an architecture for processing MCS queries in smart cities using an approach based on models at runtime that is integrated as part of an existing smart city platform called InterSCity (Del Esposte et al., 2017). The proposed approach fulfills the following goals: (a) processing

user-defined MCS queries; (b) providing a scalable MCS service; (c) dynamically adapting query processing to changes in the environment, by adjusting the set of selected devices and sensors and by allowing users to alter queries on-the-fly (mainly in the case of long-running queries); and (d) providing composite resources (based on the dynamic combination of crowd-based resources) that applications can use in a transparent way.

In order to demonstrate the feasibility of this approach, we present a scenario for monitoring the noise levels of a city in order to identify critical areas with high levels of noise. Data gathered in this way can be used by environmental control applications (Zappatore et al., 2016).

The rest of the paper is organized as follows. Section 2 presents the state of the art on MCS, a model-driven approach for MCS and its integration with Smart Cities. Section 3 discusses smart city platforms in general and the InterSCity platform in particular. Section 4 describes the proposed architecture, while its implementation is presented in Section 5. Section 6 presents a scenario to demonstrate the functionality of the platform, and Section 7 reviews the main contributions and discusses future work.

2. Mobile CrowdSensing (MCS)

MCS refers to the opportunistic or participatory use of a large set of sensors embedded in current general purpose mobile devices for the purpose of measuring and mapping interesting phenomena by means of the collaborative sharing of sensors (Ganti et al., 2011). MCS environments encompass a variety of applications that need to communicate and exchange data. The major challenges are related to the amount and diversity of devices, the dynamism of the scenarios, and the proper selection of devices to fulfill a given request.

Existing platforms for MCS address challenges such as facilitating application development, supporting efficient and scalable dissemination of sensor data, enabling mobility management of the applications and providing incentives for participatory sensing. However, the programming models in most of these platforms makes it difficult to develop dynamic applications that need to change quickly in the face of changes in the application or its environment. The next section addresses a robust alternative to address this issue.

2.1. Model-Driven Approach for MCS

A model-driven approach to MCS is motivated by the need to overcome the adaptability challenges due to the variety of applications and the mobility of devices. The use of models@runtime in MCS allows the description of the crowdsensing behavior of an application in a dynamic way, thus enabling runtime adaptation of such behavior. In general, the use of a model-based approach enables the shortening of the semantic gap between the problem to be solved and the platform being used, promoting the use of abstractions that are closer to the problem domain.

In this context, CSVM (CrowdSensing Virtual Machine) (Melo, 2014) is a platform driven by models@runtime (Blair et al, 2009) that enables the creation and execution of MCS queries by specifying and interpreting models described in a domain-specific modeling language called CSML (CrowdSensing Modeling Language).

CSVM was implemented as a distributed architecture containing 5 layers and comprised by a central component (CSVMProvider) and a distributed component (CSVM4Dev) which is instantiated on each participating mobile device. In addition to its reliance on modeling techniques, CSVM demonstrates the flexibility in creating queries from high-level models that can be modified at runtime.

CSML is the domain-specific modeling language (DSML) interpreted by CSVM. It allows the creation and manipulation of models that describe queries and their execution. Its constructs are used to model the two major functionalities required by MCS applications, namely device registration, which integrates the device as part of the crowdsensing environment, and query specification, which allows the user to create queries that involve sensor data gathered from multiple devices. These two functionalities are specified in the form of two kinds of submodels, also called schemas: Control Schema (CS) and Data Schema (DS). CS are models that represent logical CrowdSensing configurations and are further subdivided into Environment Control Schemas (ECS) and Query Control Schemas (QCS). The constructs used to specify schemas are defined in the CSML metamodel, which in turn is defined according to OMG's metamodeling architecture, the Meta-Object Facility (MOF) (OMG, 2008).

An ECS describes the crowdsensing environment and serves as a representation of the devices (and sensors) that are available. A QCS is a model at runtime that specifies one or more queries in terms of the desired types, quantities, and location of sensors, as well as the operation to be executed on sensor data (e.g., average, sum, etc.) and the type of notification of sensor data to clients (e.g., following an event-driven approach). As an example, a QCS can be used to describe a query to monitor the noise level in a specific place or region. Finally, a DS is a model that represents an empty form, which specifies the type of sensor information required in a query.

CSML can be used to specify MCS functions in different domains, including Smart Cities. Its constructs are based solely on elements of the MCS technical domain, making it independent of any specific application domain. In this work we are interested in investigating the benefits of CSML's model-driven approach to support MCS applications in the domain of Smart Cities.

2.2. MCS for Smart Cities

MCS fits naturally in smart cities since every citizen, with their mobile phones equipped with a variety of sensors, can be considered a data source in the city. Cooperation between citizens that are part of a crowd enables large-scale sensing tasks. In this context, models architectures are proposed, aiming at a horizontal approach (Petkovics et al., 2015) or even a reference architecture with shelf components also called off-the-shelves (Diniz et al., 2015).

Various platforms for Smart Cities try to incorporate MCS in order to provide a more complete platform solution that involves community (human) and collaborative sensors. Examples are CrowdOut (Aubry et al., 2014), Borja e Gama (Borja et al., 2014) and SOFIA (Filipponi et al., 2010). CSVM in turn is a complete platform to model and process MCS queries, handling the major requirements of the MCS domain. Its strength lies in the use of a model-driven approach to collect, process and store the data from devices in addition to supporting the construction of MCS applications through the use of dynamic models. These features naturally fit in the kind of dynamic environment that is characteristic of smart cities. However, CSVM's architecture does not consider its integration with other services that are required to handle the requirements of smart cities. Examples are the integration of infrastructure sensors and social networks as data sources, as well as the processing of big data that arises from the collection of data from a large number of sources.

3. Platform for Smart Cities

A smart city platform must integrate multiple domains into a complete and consistent middleware solution, providing facilities for the development, integration, deployment, and management of applications (Santana et al., 2017). Building such platforms involves challenges: enabling interoperability between a city's multiples

systems, guaranteeing citizens' privacy, managing large amounts of data, supporting scalability, supporting adaptability of dynamic environment, and dealing with a large variety of sensors. In order to overcome these challenges, several smart city platforms have been developed, such as OpenIoT (Solatos et al., 2015), SMARTY (Anastasi et al., 2013), U-City (Piro et al., 2014), and InterSCity (Batista et al., 2016), which was chosen by this work to present a microservice architecture that allows easy adaptation of new services like MCS and it is discussed next.

3.1 InterSCity

The InterSCity platform has a microservice-based architecture designed as a unified reference architecture for smart cities (Santana et al., 2017). The architecture is shown in Figure 1 as a set of high-level cloud-based (RESTful) services. To provide easy and decentralized communication, each InterSCity microservice has well-defined boundaries to communicate with both IoT devices and smart city applications. Currently, the platform is composed of six microservices that provide: integration with different IoT devices (Resource Adaptor), data and resource management (Resource Catalog, Data Collector and Actuator Controller), resource discovery through context data (Resource Discovery), and graphical interface for visualization (Resource Viewer).

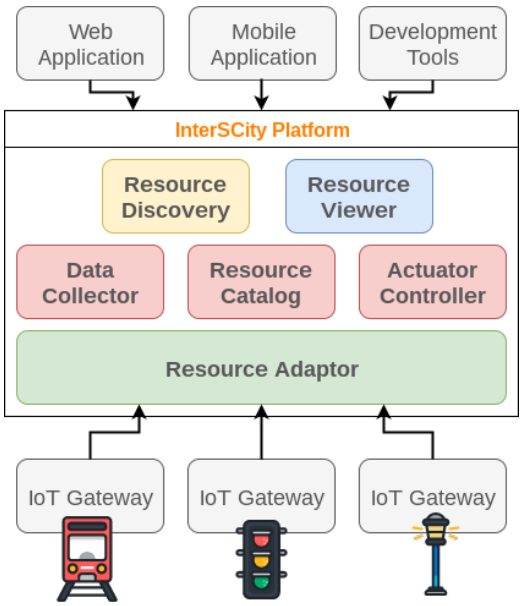

Figure 1. The InterSCity Platform Architecture (Del Esposte et al., 2017).

All microservices are implemented with REST APIs for synchronous messaging over HTTP and RabbitMQ of the Advanced Message Queuing Protocol (AMQP) for assynchronous calls. In addition, another important aspect of the platform is the mapping of each physical entity that makes up the city (cars, buses, lampposts, traffic lights, etc.) to a logical resource. These resources comprise attributes (e.g., location and description) and functional capabilities to provide data and receive commands.

Some design principles were considered when building the InterSCity platform, with emphasis on scalability (in terms of the number of devices, users and components, and volume of city-related data) and evolvability (very dynamic urban environments tend to change constantly in terms of organization, regulations, problems, opportunities and challenges). However, regarding the support for MCS applications (queries) and resources, the InterSCity architecture has two limitations: (1) lack of runtime adaptation of query processing; and (2) lack of transparent support for composite resources (crowd/group of sensors). This paper addresses these limitations with an extension of the

InterSCity platform to support the processing and management of MCS applications through a model-driven approach.

4. MCS Architecture

In general, the architecture of smart city platforms must include components to support the construction of applications, manage and communicate with city network nodes, integrate with existing social networks, store and manage the collected data, and capture context variations and adapt to it (Santana et al., 2017). In line with this generic approach, we propose an architectural extension of InterSCity to support construction and processing of MCS queries according to the model-driven approach used in CSML.

The proposed architecture, shown on the left part of Figure 2, comprises all components already implemented in InterSCity, augmented with a new microservice called CrowdSensing Engine, responsible for processing MCS queries and described next.

4.1. CrowdSensing Engine

The CrowdSensing Engine is a microservice responsible for processing MCS queries. For the construction of this microservice the InterSCity design principles were maintained so that the extension does not compromise the original structure. As such, this component was developed according to the evolution requirements of the platform, maintaining the characteristics of microservices in a way that is weakly coupled, scalable and has well-defined interfaces for external communication. The CrowdSensing Engine has a five internal components, as shown in Figure 2 and described next.

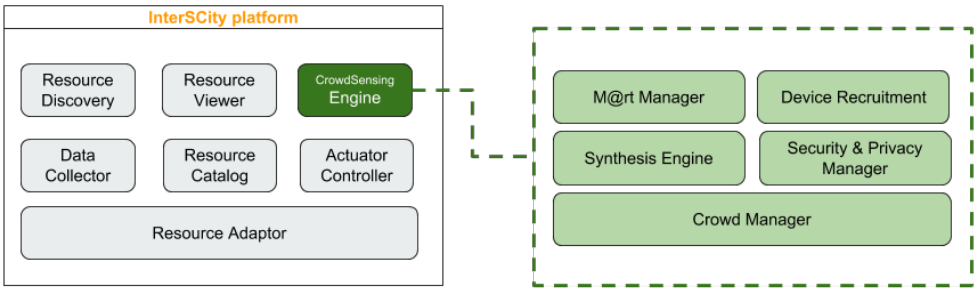

Figure 2. The InterSCity CrowdSensing Engine (left) and its internal components (right).

Crowd Manager. This component is responsible for keeping the crowd always up to date. A crowd represents the set of devices recruited to provide data for a query. Crowd Manager manages the status of such devices to monitor for failures and availability, in which case it interacts with the Device Recruitment component to select another device. In addition, it is also responsible for registering the crowd in the Resource Catalog as a logical resource with a specific capability (e.g., temperature, humidity etc.) making its data available to other applications (not necessarily crowdsensing ones). Note that each query can generate a different crowd, all of which are managed by this component.

Each crowd has a CSML model that represents it and is maintained at runtime, so that changes identified by Crowd Manager trigger commands for the M@rt Manager to update the model. This model is based on the query description in CSML (more specifically a user-generated QCS) and composed of the recruited devices.

M@rt Manager. This component keeps the runtime model up to date. This includes all aspects of the MCS environment, notably query models and crowd models. More specifically, it manages adaptation rules, so that when a rule is triggered, it

communicates the event to the Synthesis Engine and Device Recruitment components for appropriate handling. An adaption rule can be triggered when a device is unavailable, in which case the M@rt Manager must change the model by removing the device and inserting a newly recruited device.

Synthesis Engine. This is the central component of the microservice and all queries must pass through it. It is responsible for interpreting all the models described in CSML and received by the microservice. It has an interface for communication with the other components inside the microservice, and provides a REST API for communication with MCS applications. Therefore, all internal communication with this component is performed through method calls and external calls are carried out via HTTP REST protocol. This is the only component that interacts directly with MCS applications.

As part of model interpretation, it parses a JSON-based input model and converts it into an internal model described in CSML; then it transforms the elements of the CSML model into HTTP commands that carry out the intent expressed in the model.

Device Recruitment. This component manages internal and specific recruitment policies to access CrowdSensing resources. It communicates with Resource Discovery to select resources according of a specific query and to construct a QCS instance. It has a direct communication interface with Resource Discovery. It functions as a broker between the CrowdSensing Engine and the set of cataloged (registered) resources.

Security and Privacy Manager. It is responsible for applying pre-defined privacy and security policies. With regard to security aspects, this component should guarantee confidentiality, availability, integrity, authenticity, non-repudiation, and auditing. To do this, it implements communication protocols that employ encryption and access control through an authentication system and access control lists (ACLs). Privacy is managed based on a set of user-defined rules (usually restrictive) associated with each user, informing access restrictions to specific sensors or by certain types of application.

The remainder of this section describes the interaction protocols that these components follow in order to process MCS queries.

4.2 MCS Query Processing

To perform CrowdSensing for an application, a platform must allow the registration of devices and the subscription (or sending) of CSML queries. **Device Registration** is performed by the generic components of InterSCity as shown Figure 3. In this process, (1) the device sends an HTTP POST command with its capabilities (sensors it wants to share), (2) Resource Adaptor sends the resource meta-data to Resource Catalog, (3) Resource Catalog publishes an event to the RabbitMQ message bus, which may notify (4) the Data Collector and Actuator Controller to inform that the resource has the specified sensor and actuation capabilities.

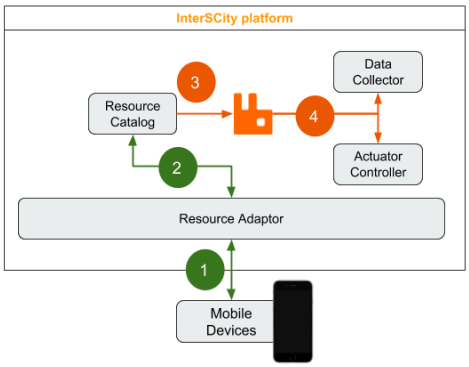

Figure 3. MCS Device Registration in InterSCity platform

Query Subscription (query processing), in turn, directly involves the CrowdSensing Engine microservices as shown Figure 4(a) and 4(b). The following steps describe how components interact during query processing. First, the application describes the query model in CSML and (1) sends the model to the platform through the REST API provided by CrowdSensing Engine (more specifically, via the Synthesis Engine component interface). The Synthesis Engine component performs the parsing, (2) sends the model to the M@rt Manager for storage, and converts the query described in CSML into commands to recruit the devices, which are then send (3) to Device Recruitment.

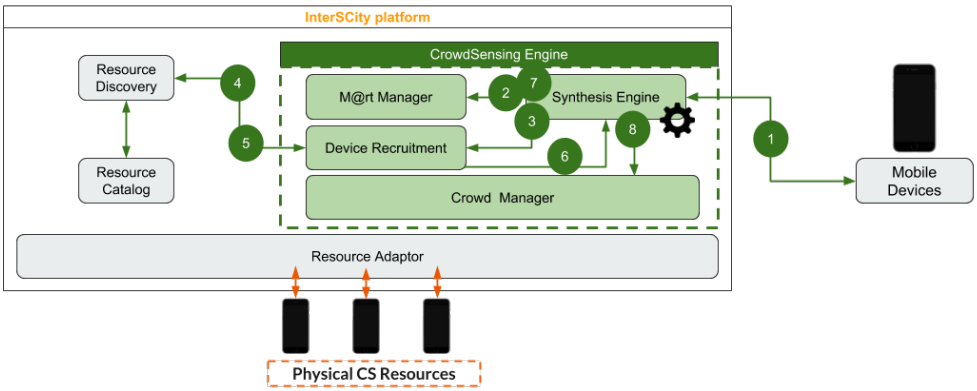

Figure 4(a). MCS Query Processing part 1

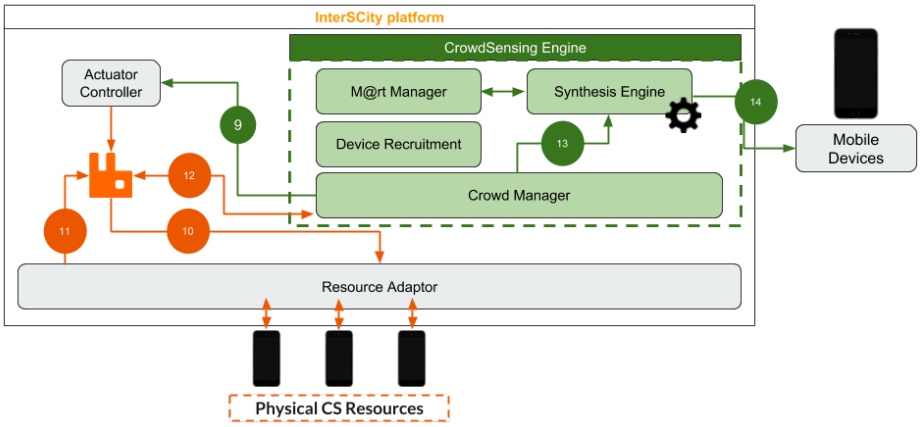

Figure 4(b). MCS Query Processing part 2

Device Recruitment applies policies to access the MCS resources and (4) sends recruitment requests to get data about the recruited resources. If devices are available in accordance with the query, Resource Discovery (5) returns a list of the devices that were recruited, identified by uuid (notation used in InterSCity to identify each cataloged resource). Synthesis Engine (6) receives the list sent by Device Recruitment in JSON, performs parsing, converts the list into a QCS instance (in CSML), (7) sends the up to date the model to M@rt Manager, and (8) sends to Crowd Manager the group of recruited devices (at this point, it creates the group/crowd and an id for it).

Crowd Manager generates and sends commands (9) to Actuator Controller to obtain current data from recruited devices and to generate asynchronous commands to get the status of devices. Resource Adaptor is notified by RabbitMQ and forwards the notification to the devices (10). After data capture, Resource Adapter (11) publishes the data obtained. Crowd Manager (12) receives the notification, consumes the message and (13) sends it to Synthesis Engine, which applies the appropriate business rules and

performs data merging (referring to M@rt Manager). Finally, (14) the response is sent to the requestor.

5. Implementation

In this section, we briefly describe some aspects of a proof-of-concept prototype of the CrowdSensing Engine microservice. The prototype comprises an implementation of the five internal components described in Section 4, which are responsible for processing CSML MCS queries. The interaction between these components are performed through remote calls to methods, as well as through trigger events.

CrowdSensing Engine was implemented in Ruby. Its implementation is encapsulated as a microservice that provides a RESTful interface to communicate with other platform microservices and external applications. For asynchronous calls, we used the RabbitMQ implementation AMQP. More specifically, for the processing of MCS queries, an `MCS' topic was created.

Like the other microservices of the platform, CrowdSensing Engine is encapsulated in its own Docker lightweight container which can be deployed and maintained independently.

6. Example Scenario

In this section we present a scenario of applicability of the proposed approach in order to demonstrate the feasibility of the solution and its ability to meet the major specific requirements for MCS in the smart cities domain. In order to demonstrate the scope of the proposal, this scenario comprises the complete cycle of processing MCS queries applied in the domain of smart cities.

The example application in this scenario corresponds to the monitoring of noise level in an urban space. One of the possible ways for handling this scenario is to offer both citizens and city managers new ways for managing environmental monitoring. Such management can be done by a smart city platform with MCS components that exploit the capabilities of the microphones embedded in citizen's smartphone's as sound sensing devices in order to create large-scale noise maps and suggest city managers suitable noise reduction interventions.

The first step is to specify CSML queries (using an application designed for this purpose) related to the problem domain, informing the sensors (type and quantity) that are required for the monitoring, as well as the location to be monitored (e.g., "*100 audio sensors located in the X neighborhood during 10 hours*"). It is also possible to specify an operation to be performed over the collected data (e.g., average). Once the query is created, it is sent to the InterSCity platform.

The second step occurs within the platform, where the CrowdSensing Engine microservice receives the request and initiates its processing by following the steps described in Section 4.2. After interpreting the query, the CrowdSensing Engine microservice sends commands to discover resources (devices) that meet the requirements specified in the query. At this point, the resources are searched in a catalog of resources maintained by the platform.

A list of 100 devices must be returned to the CrowdSensing Engine to update the query model, which will include the sensor type, location, query duration, and devices that have been recruited. If any of the devices fail or if the user changes the query parameters during its execution, the CrowdSensing Engine identifies the fact and initiates the adaptation process. This process involves updating the model by removing the failed device and/or applying the new requirements stipulated by the user as changes in the model kept at runtime.

Finally, the data obtained are processed and the results are returned to the user, informing in this scenario the noise level of that specified region.

7. Conclusion and Future Work

The development and deployment of smart city platforms faces a number of challenges such as privacy, data management, heterogeneity, communication, scalability, city models and dynamism. Through these challenges there is an increasing need for smart city solutions that take advantage of the latent potential of existing open source platforms.

In this paper, we presented an architecture for processing MCS queries in the smart cities domain. More specifically, we propose an extension of the InterSCity platform to support CrowdSensing applications through the development of a microservice based on a models@runtime approach for MCS. The approach uses the CSML modeling language to model crowdsensing queries, thus supporting the development of applications in this domain.

The main contribution this work is the demonstration of feasibility of integrating MCS in the domain of smart cities using a models@runtime approach. In this way, the approach proposed in this article makes it possible to opportunistically (or in a participative way) use the latent potential of sensors embedded in smartphones through the implementation of the Mobile CrowdSensing paradigm as part of a platform for smart cities. This paradigm also allows to combine different resources in order to generate more precise and useful information for the applications. In addition, this work also includes the use of models@runtime to allow real-time adaptation and improve the modeling of city aspects (through MCS query modeling in CSML) from which it differs from the other platforms mentioned in Section 3.

Our ongoing work includes the performance of experiments to evaluate the scalability and performance of the proposed microservice. For this evaluation, we will consider the time spent by each internal component of the microservice, in order to identify potential bottlenecks and propose improvements in its implementation.

Acknowledgement

This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq, proc. 465446/2014-0, CAPES proc. 88887.136422/2017-00, and FAPESP, proc. 2014/50937-1 and proc. 2015/24485-9.

References

Alvear, O., et al. "Crowdsensing in Smart Cities: Overview, Platforms, and Environment Sensing Issues." Sensors 18.2 (2018): 460.

Anastasi, Giuseppe, et al. "Urban and social sensing for sustainable mobility in smart cities." Sustainable Internet and ICT for Sustainability (SustainIT), 2013. IEEE, 2013.

Aubry, E., Silverston, T., Lahmadi, A., & Festor, O. (2014, March). “CrowdOut: a mobile crowdsourcing service for road safety in digital cities.” In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on (pp. 86-91). IEEE.

Batista, D. M., Goldman, A., Hirata, R., Kon, F., Costa, F. M., & Endler, M. "Interscity: Addressing future internet research challenges for smart cities." Network of the Future (NOF), 2016 7th International Conference on the. IEEE, 2016.

Blair, G.; Bencomo, N.; France, R. B. “Models@ run.time.” Computer, v. 42, n. 10, 2009.

Borgia, E. “The Internet of Things vision: Key features, applications and open issues.” Computer Communications, v. 54, p. 1-31, 2014.

Borja, R., & Gama, K. (2014). “Middleware para cidades inteligentes baseado em um barramento de serviços. ” X Simpósio Brasileiro de Sistemas de Informação (SBSI), 1, 584-590.

Celino, I., Kotoulas, S. “Smart Cities [Guest editors' introduction].” IEEE Internet Computing, v. 17, n. 6, p. 8-11, 2013.

Del Esposte, A. D. M., Kon, F., Costa, F. M., & Lago, N. “InterSCity: A Scalable Microservice-based Open Source Platform for Smart Cities”, 6th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS), Porto, Portugal, 2017

Diniz, H. B., Silva, E. C. G. F., and Gama, K. "Uma Arquitetura de Referência para Plataforma de Crowdsensing em Smart Cities." XI Brazilian Symposium on Information System. 2015.

Filipponi, L., Vitaletti, A., Landi, G., Memeo, V., Laura, G., & Pucci, P. (2010, July). “Smart city: An event driven architecture for monitoring public spaces with heterogeneous sensors.” In Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on (pp. 281-286). IEEE.

Ganti, R. K.; YE, F.; LEI, H. “Mobile crowdsensing: Current state and future challenges.” Communications Magazine, IEEE, 49(11):32–39, 2011

Melo, P. C. F. "CSVM: Uma plataforma para crowdsensing móvel dirigida por modelos em tempo de execução." (2014).

OMG, Q. V. T. “Meta object facility (mof) 2.0 query/view/transformation specification.” Final Adopted Specification (November 2005), 2008.

Petkovics, A., et al. "Crowdsensing solutions in smart cities: Introducing a horizontal architecture." Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia. ACM, 2015.

Piro, G., et al. "Information centric services in smart cities." Journal of Systems and Software 88 (2014): 169-188.

Santana, E. F. Z., Chaves, A. P., Gerosa, M. A., Kon, F., & Milojicic, D. S. (2017). Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. ACM Computing Surveys (CSUR), 50(6), 78.

Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J. P., Riahi, M. & Skorin-Kapov, L. (2015). Openiot: Open source internet-of-things in the cloud. In Interoperability and open-source solutions for the internet of things (pp. 13-25). Springer, Cham.

Stojanovic, D., Predic, B. and Stojanovic, N. "Mobile crowd sensing for smart urban mobility." European Handbook of Crowdsourced Geographic Information (2016): 371.

Zappatore, M., Longo, A., & Bochicchio, M. A. Using mobile crowd sensing for noise monitoring in smart cities. In Computer and Energy Science (SpliTech), International Multidisciplinary Conference on (pp. 1-6). IEEE, 2016.