An Extensible Platform for the Treatment of Heterogeneous Data in Smart Cities

Cícero Alves da Silva and Gibeon Soares de Aquino Júnior

Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte
Natal, RN, Brazil
Email: cicerojprn@gmail.com, gibeon@dimap.ufrn.br

***Abstract*—Nowadays, there is a lot of devices of varying technologies in the urban environment, which makes the integration of data generated by them a difficult process due to their heterogeneity. However, it is important to manage these data in an integrated way to enable the exchange of information between existing fields and assisting in the decision-making process. Moreover, there is no way to tell how these data will need to be processed since each application may require it to be available obeying specific processes. Thus, this article describes the design and implementation of a platform that aims to integrate, process and make available data streams from heterogeneous sources. It also defines an extensible data processing flow, which makes the creation of new processes for existing data and the inclusion of new types of data easier. Finally, a case study was conducted, which used a parking lot as scenario and assessed extensibility and performance aspects related to platform implementation.**

Keywords–Smart Cities; Software Architecture; Extensibility.

I. INTRODUCTION

The widespread use of intelligent devices and other types of sensors resulted in the emergence of the Internet of Things (IoT), a paradigm in which the objects of the everyday life are equipped and able to communicate with other objects and users, which makes them a part of the Internet [1]. Thus, these objects are able to work in different urban environments, providing data that are collected in them and enabling the Smart Cities concept to be used. Even though the term "Smart City" has been widely used nowadays, it does not present a standardization regarding its meaning. However, it is known that a smart city should pay special attention to performance improvement in six different areas: Economy, People, Governance, Mobility, Environment and Living [2][3].

Nonetheless, only the use of these objects is not enough to improve urban life [4]. It is important that the management of the data generated in them is carried out in the same place, allowing the exchange of information between the existing sectors to happen and assisting in the decision-making process. However, this data integration is not a trivial task because of the devices' heterogeneity [1][5][6], since they use different technologies and different communication protocols and produce data flows with multiple formats and different characteristics.

Furthermore, applications that consume these data may require them to be made available in different forms, making it necessary for them to be processes before its delivery. Therefore, to assist in the comprehension of the complied problems, the systems that propose to process data flows from these heterogeneous sources need to filter them, combine them and assemble them, thus producing new data as output [7]. However, there is no telling in which form the data needs to be processed, since the same data may need to be processed in different ways to meet the application's needs and since there may also be the need to perform the inclusion of new types of data in the platform.

Finally, this article discusses the definition, design and implementation of a Smart City platform whose focus is related to the integration, prossessing and availability of data flows from heterogeneous sources in an urban environment. In addition, this study also discusses the process of extensible data processing defined in this platform, which allows the data to be processed according to its specific characteristics and the application's needs. Section II discusses a few related works. Subsequently, Section III shows the platform proposed in this study. Section IV, in turn, discusses a case study that used a parking lot scenario and assessed some important aspects related to the implementation of the platform. Finally, Section V shows this study's conclusions and future work.

II. RELATED WORK

In the study of Anthopoulos and Fitsilis [8], a research is held in smart cities in order to develop an architecture to be used in the management of urban services. However, unlike the present study, Anthopoulos and Fitsilis' work deals only with the architecture's description. Thus, it is not possible to identify the modules that must be implemented for the proposed layers to work and it is not possible to assure that these layers are effective to work with the data generated in the urban environment.

In Filipponi et al.'s work [9], an event-based architecture that allows the management of heterogeneous sensors to monitor public spaces is presented. However, this architecture is different from the one proposed in this work, since its use is very limited and it does not incorporate many requirements such as privacy and monetization.

The MAGIC Broker 2 platform, which focuses on objects' interoperability and proposes to work in an IoT environment, is presented in Blackstock et al.'s article [10]. However, the authors state that this platform is not ready to work in a Smart City environment. In contrast, in the present work, the platform is designed precisely to deal with this area of study.

Middlewares for IoT are proposed in Gama, Touseau and Donsez's study [6] and in Valente and Martins' study [11]. However, they differ from the platform proposed in this study since they do not perform the extraction of knowledge from the integrated data and do not have privacy strategies for the transfered information.

Andreini et al.'s article [12] discusses an architecture based on the principles of service orientation. However, it is limited to smart objects' geographical location issues. Furthermore, it does not address data privacy and does not allow the aggregation and extraction of the knowledge found in them.

III. PROPOSED PLATFORM

The platform proposed in this paper aims to enable integration, processing and availability of different types of data generated by the sensors that exist in the urban environment. Furthermore, it focuses on providing the extensibility of processing tasks performed on these data due to the fact that it is not possible to predict in what form their flows need to be processed.

Thus, the extensibility of the processing steps is important due to the fact that the applications are so dynamic and may require different processing ways for the same data flow and due to the fact that with time, new types of data will turn up as a result of the emergence of a new source.

This way, moving from the intended goals to the platform and seeking to provide this extensible data processing feature, the following requirements were defined:

1) Data retrieval from sources with heterogeneous technologies;
2) Availability of data integrated into the platform;
3) Data association allowing information from different domains to be combined to work in an unified manner;
4) Follow the modular and "pluggable" approaches, making the maintenance and extension of the platform easier;
5) Have a well defined data transformation process, since its modules represent specific stages of processing, which makes it necessary for them to have specific responsibilities within the platform. The data transformation process should be extensible so that the processes can be suitable to work according to the characteristics of each type of data;
6) Keep the transmitted data's privacy;
7) Allow the extraction of knowledge from large volumes of data integrated to the platform;
8) Enable monetization, allowing the developers of the services to sell the data created in them.

Table I shows how the studies analyzed in Section II treat the requirements listed above for the proposed platform. Thus, it is clear to see that none of them defines an extensible transformation flow for processing the data in their architecture proposals, which is a requirement that is the main focus of this proposal. In this flow, we determine the steps required to process a group of data that is integrated to the platform in order to deliver them in the best way possible to be used in the development of new systems. Moreover, the extensibility of the defined steps allows these steps' processing are realized according to the characteristics of each type of data.

TABLE I. REQUIREMENTS ATTENDED BY THE RELATED WORKS.

Requirement	Works
Retrieve data from heterogeneous sources	[12], [8], [10], [9], [6], [11]
Create new services	[12], [8], [10], [9], [6], [11]
Support data aggregation	[8], [9], [6], [11]
Well-defined and extensible data processing	-
Allow the extraction of knowledge	-
Modular approach	[10], [9], [6], [11]
Pluggable approach	-
Maintains data privacy	[8]
Monetization	-

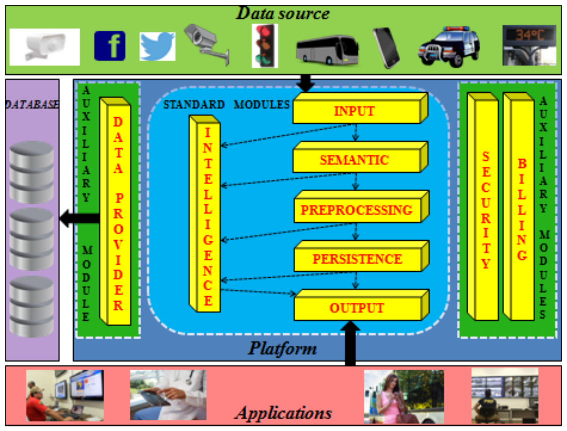

Figure 1. Proposed platform.

A. Architecture

Due to the requirements listed in Section III, we decided to carry out the implementation of the proposed architecture in the Open Services Gateway Initiative (OSGi) framework. This technology makes the development of modular Java softwares easier due to the fact that it provides many benefits related to manageability and maintainability [13], which is essential for this solution since the extensible transformation process thought for it aims to use the modular and "pluggable" approaches, allowing the extensions to be easily inserted and removed.

Figure 1 displays the platform proposed in this article. In Figure 1, we can see that it was planned in a way to support data from different sources, which are treated within it and then are made available, allowing the creation of new applications. In addition to this, it is possible to identify that when it comes to the database that should be used, it is flexible and supports the use of different types of database. The platform also has a set of **standard modules**, which are responsible for defining the steps of the extensible processing flow. Moreover, in the proposed solution, there is a set of **auxiliary modules** that increase the features that are important to it.

Each of these standard modules has its own specific responsibilities in the architecture and provide its basic behaviors. Furthermore, as shown in Figure 2, they are responsible for defining the extension points, allowing different implementations to be generated and "plugged" on to the platform.

The **specific modules** are responsible for implementing the processing tasks for each step of the extensible processing flow. Therefore, to add a source to the architecture, it is necessary to implement the specific modules that are able to handle the

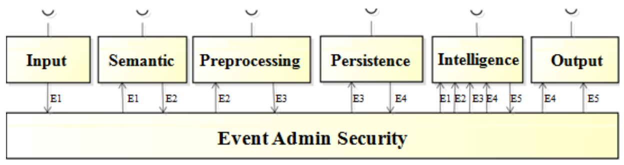

Figure 2. Extensible architecture.

type of data being inserted and then plug them to their relating standard modules.

In Figure 2, it is also possible to identify the existence of the **Event Admin Security** module, which is an extension of the Event Admin available in OSGi. This extension was carried out with the goal of adding an additional security requirement related to the access to messages transferred in this module. Thus, this capability ensures that only the architecture's standard modules can receive messages from the Event Admin, preventing the specific modules to interfere in their flow.

The standard modules work partly in a similar way and are only distinguished from each other in the processing step for which they are responsible. In general, a standard module receives a set of data. Then, it checks the specific modules that are interested in the type of data received and passes it to those who are allowed to access it. Thus, these specific modules process and return the data to the standard module which, finally, publishes it using the **Event Admin Security**. This way, the architecture has six standard modules, which are:

1) **Input**: the modules that are "plugged" on to the Input are responsible for integrating different data sources that exist in the cities to the proposed architecture;
2) **Semantic**: is responsible for receiving the data that is integrated in the Input and representing them in the format that the developer feels is most appropriate to the system that is being implemented;
3) **Preprocessing**: receives the data treated in Semantic and is responsible for filtering it;
4) **Persistence**: its tasks is to receive the preprocessed data and the primary responsibility of the modules that are "plugged" on to it is to store the received data in the architecture;
5) **Intelligence**: is responsible for receiving all of the data processed by the modules mentioned above. This way, the specific modules process this data and when their algorithms can identify any relevant knowledge, event 5 (E5) is published;
6) **Output**: is responsible for receiving the data processed by the Persistence and the Intelligence modules. Finally, each individual Output module provides access to the architecture's data, allowing new applications to be developed.

The proposed architecture also has three auxiliary modules: **Security**, which implements the policy of permissions to access data that are transferred within the architecture; **Billing**, which is responsible for accounting the messages that are accessed by the specific modules to enable later billing related to data consumption; and **Data Provider**, whose function is to carry out the management of the stored data and allow them to be accessed by the specific modules.

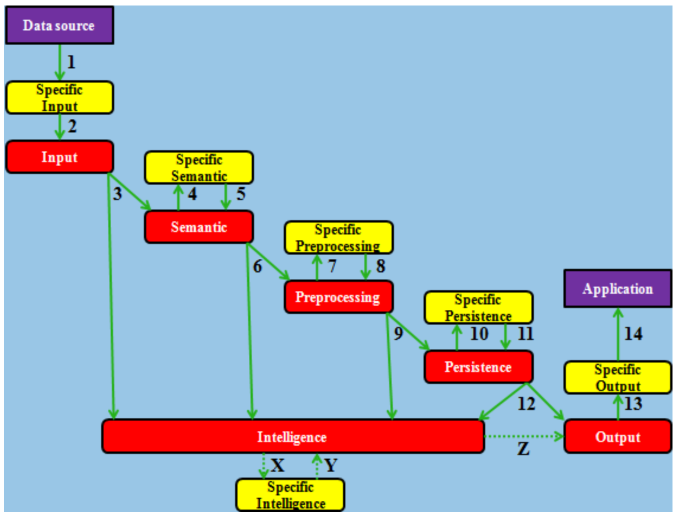

Figure 3. Steps of the data processing flow.

To perform the data aggregation process in a module, it is only necessary for it to have the set of permissions to access data from different services that lead to the compound service. Finally, the developer is not obliged to provide specific modules for all of the steps in the transformation flow. Thus, the platform will transfer the data to the next module of the flow when it is not possible to find, in a standard module, specific implementations responsible for working with the data type that was received.

B. Data flow in the platform

As defined in Figure 3, by making use of these six modules and using a simplified scenario where there is only one specific module "plugged" to each standard module, the basic extensible flow occurs through a set of 14 steps. First, the data is sent from the source to the **Specific Input** module responsible for receiving it (Step 1), which, in Step 2, forwards it to their relating standard module.

Then, in Step 3, the data is transferred from the Input to the next step (Semantic). Thus, this standard module forwards it to the specific module that is capable of working with it (Step 4). After it is received, the **Specific Semantic** module performs the first data transformation, since it is at that instant that it starts to be represented in the format chosen by the developer of the specific system. After that, in Step 5, ir is sent back to Semantic and then the data is published by this standard module (Step 6).

Preprocessing receives the data transferred in Step 6 and delivers it to the **Specific Preprocessing** module (Step 7), which performs a filtering process in which the set of data is subjected to a cleaning and selection process. After that, in Step 8, the filtered data returns to the Preprocessing module, which passes it forward (Step 9).

In Step 10, Persistence transfers the pre-processed data to the **Specific Persistence** module. After that, this specific module performs the data storage process and then, in Step 11, returns the last state of the problem's data to Persistence, which forwards it to the next step of the flow (Step 12).

Subsequently, Output passes the data to the specific module

(Step 13). Thus, in Step 14, the **Specific Output** module makes the data accessible to applications.

It is important to note that any of the specific modules can perform data aggregation in its processing tasks. In addition, Figure 3 also shows a knowledge discovery flow. In it, the Intelligence module receives all of the data delivered in steps 3, 6, 9, and 12 of the basic flow. Every time a set of data is received, the standard Intelligence module forwards it to the **Specific Intelligence** module (Step X) which processes it at all times in an attempt to identify any knowledge relevant to the problem. Therefore, when something meaningful is identified, the Specific Intelligence module returns the information to the standard Intelligence module which, in its turn, sends it to the Output module, which makes them available to the applications. Finally, for this flow, the letters X, Y and Z were used since it is not possible to predict the moment in which every one of the steps will be executed in the processing flow because they do not follow a sequential execution like the basic flow does.

IV. Case Study

This section describes a case study that aims to evaluate two behavioral aspects of the implemented platform: the easiness of the creation of specific modules (**Extensibility**) and the data processing capacity (**Performance**). Finally, the planning process and its description followed the guidelines set forth in [14][15][16].

A. Planning

This case study investigates the following research questions (RQ):

- **RQ1**: Is the platform extension process that is carried ou through the development of specific modules a simple activity?
- **RQ2**: Is the performance of the data flow's treatment process impaired in any way due to the existence of a set of steps for information processing?
- **RQ3**: Is the performance of the data flow's treatment process impaired when specific modules plugged to standard module are used?

The **subject** who used the platform that was proposed and implemented in this work was a developer with experience in the development of Java and OSGi applications. Moreover, the used **object** was an extension of the proposed platform depeloped to integrate data from a parking lot. Thus, in this scenario, we intended to access data from the server that stored the parking lot's information, process it using the platform's extensible flow and then make it available for the development of new applications.

The analysis units for this case study are: the implemented platform and its extension that enables to work with the parking lot's data. Thus, the platform's standard modules were evaluated regarding the performance of the data flow's transformation process. The extension used to handle the parking lot's resource, in its turn, was explored regarding the extensibility analysis and the evaluation of the performance of the data flow when specific modules are "plugged" on to standard modules.

B. Execution

To insert the data from the parking lot in the proposed platform, the implementation of specific modules to work with this source was generated. A priori, the additional module **Parking Model** was developed, which is used by all of the specific modules and whose responsibility is to mold in classes the data from the parking lot source.

Then, the **Parking Input** module was implemented, which integrates the data generated in the parking lot to the platform. Subsequently, the **Parking Semantic** was generated, which is responsible for representing such data in objects. Thereafter, the **Parking Preprocessing** module was developed, whose duties are to eliminate data duplication and select only the main data of the problem. In sequence, the **Parking Persistence** was implemented, which is only responsible for performing the received data storage step. In this study, the **Parking Intelligence** module was also developed, which only stores in a file the logs from all of events 1, 2, 3 and 4 sent in the **Event Admin Security**. This was important to confirm the sequence of the sent events. The **Parking Output** module is a Representational State Transfer (REST) module that works as a gateway for the parking lot's resource data processed in the architecture.

After the implementation of all of these modules, we moved on to the stage of evaluation of all of the **extensibility** and **performance** aspects of the platform. This evaluation was performed in a machine with Windows 8.1 operation system Single Langue 64-bit, Intel (R) Core (TM) i3-3227U CPU @ 1.90GHz and 3.87 GB of RAM processor.

Regarding the extensibility, we collected the amount of lines of code implemented in each of the specific modules of the parking lot's system, as shown in Table II. On this count, all of the lines of code in the source code's file were accounted for, including imports, statements, etc. In addition, we also counted the lines of code that are directly related to tasks that are necessary to "plug" these modules to the platform, as shown in Table III.

TABLE II. Lines of code of parking lot's system

Number of lines of code	
Parking Input	78
Parking Semantic	66
Parking Preprocessing	79
Parking Persistence	68
Parking Intelligence	78
Parking Output	61
Parking Model	56
Total	486

TABLE III. Number of lines of code in the parking lot system's modules (ignoring statements and general code)

Number of lines of code	
Parking Input	7
Parking Semantic	9
Parking Preprocessing	8
Parking Persistence	10
Parking Intelligence	9
Parking Output	12
Total	55

Regarding the performance, this requirement was evaluated using the standard of measurement of the time it takes for the data to be transferred in the flow. This measure was calculated based on the time required for a message to be transferred from the input point to the end of the processing flow. For this purpose, the average time that it takes a certain amount of packages sent at once to go through all of the processing steps of the platform was calculated. In addition, for each amount of packages sent, ten samples were collected and their general average time was generated using (1).

$$a = \frac{\sum_{j=1}^{j=10} \frac{\sum_{i=1}^{i=p} ti}{p}}{10} \qquad (1)$$

Where:

- ***a*** – represents the general average;
- ***p*** – represents the amount of packages received;
- ***ti*** – represents the transfer time for the ***i*** package.

C. Threats to validity

For the case study, four types of validity were evaluated:

- **Construct validity**: data capture for this case study's execution was performed using quantitative surveys related to factors analyzed for the implemented platform. Moreover, this process took place in a single machine, preventing changes in computer settings to compromise the values collected in the study;
- **Internal validity**: the features of the subject that performed the case study decreased the risk that factors related to inexperience in the development of Java and OSGi-based applications got out of control;
- **External validity**: programmers that are beginning to work with Java and OSGi can generate solutions with a larger amount of lines of code than those developed by the subject that performed the case study. Finally, the use of computer settings that are different from those specified in Section IV-B will influence the time it takes for messages to be processed by the platform;
- **Conclusion validity**: quantitative data that contributed to the platform evaluation process were used. Regarding performance data, they were collected in several samples in order to get an average, preventing that deviations that reflected only one specific time influence the outcome.

D. Answers to the research questions

This subsection answers the research questions raised in Section IV-A.

*1) **RQ1**:* The extension of the modules responsible for processing the data flow is a simple task, since it is only necessary to implement a small part of the code in order to plug them to the platform. As shown in Table II, in order to carry out the specific application of the six modules of the parking lot system, the implementation of 486 lines of code was necessary. However, by observing Table III, it is possible to note that less than 1/8 of the lines accounted for in Table II are directly responsible for providing the extension process defined in the standard modules.

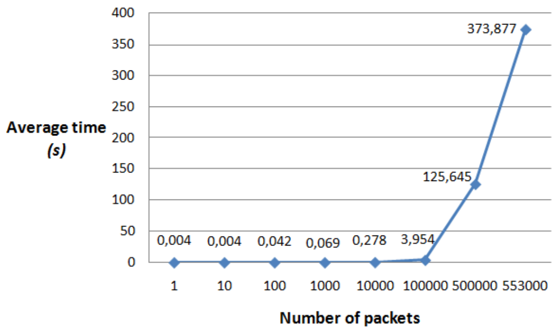

Figure 4. Average time to transport messages depending on the amount of packages (512 MB limit)

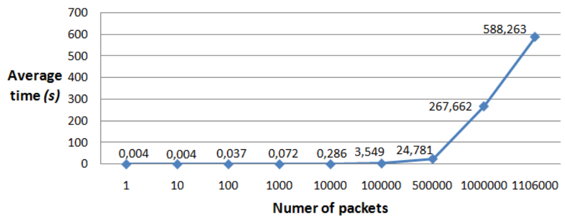

Figure 5. Average time to transport messages depending on the amount of packages (1,024 MB limit)

*2) **RQ2**:* By looking at the flow's transfer data, it is possible to realize that the data processing steps do not affect significantly the platform's performance. However, this feature depends on the settings of the computer in which it runs since, as shown in the results in Figure 4 and Figure 5, it is possible to note that the average transfer time for up to 100,000 packages received at the same time is stable, but when this number of messages is increased, the transfer time increases dramatically. Another factor that can prove this fact is the moment in which the memory limit was doubled. With this, the transfer time for the amount of 500,000 packages went down to 80.27% when compared to the 512 MB of RAM memory experiment. Furthermore, by providing the platform twice the RAM memory, the maximum number of packages supported also doubled, which shows that the amount of messages supported by Event Admin Security depends on the amount of memory available.

*3) **RQ3**:* By analyzing the graph shown in Figure 6, it is possible to note that the use of specific modules to perform the processing of the type of messages in the parking lot data causes a loss of performance in the delivery of packages when compared to the experiment shown previously in Figure 4. However, the average time only reaches very high values when the amount of messages received simultaneously is also very high, as can be seen in Figure 6, where up to the amount of 1,000 messages, the average transfer time is approximately 2 seconds.

V. Conclusion and Future Work

This article shows the details in definition, design and implementation of a platform that aims to integrate, transform and provide heterogeneous data generated in the urban environment. It has an extensible processing flow, which is important because it is not possible to predict how the different

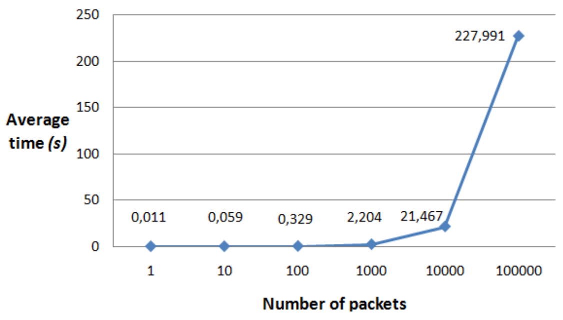

Figure 6. Average time to transport messages depending on the amount of packages using the parking lot's specific modules (512 MB limit)

types of data need to be processed in order to be delivered to the applications and because as time goes by, new types of data will emerge and will also need to be ”plugged” to the platform. This way, the main contribution of this work and of the proposed platform was the creation of an extensible processing flow that allows data processing to be suitable to work according to the characteristics of each type of data existing in the ecosystem.

Through the extension that was performed in order to work with the parking lot scenario, it was possible to insert data from a first source into the platform. With the specific modules that were developed, it was possible to test the extensible processing flow, wherein they perform processing according to the characteristics of the parking lot's feature data. Furthermore, it was possible to attest the operation of the standard modules and the auxiliary modules defined in this work.

With the case study, it was possible to evaluate two important aspects related to the platform implementation proposal. The extensibility characteristic was a process that was easily carried out due to the fact that the standard modules made available the interfaces that define the behaviors associated with it, which makes the process of extension and “plugging” specific modules easy. Moreover, with the evaluation of the performance, there is a proof that the steps defined do not burden significantly the transfer of messages in the platform.

As future work, we intend to perform further case studies of the use of the platform, in which we aim to work with scenarios where there are different sources of data allowing their aggregation and also the development of multiple applications. Furthermore, there is the intent to evaluate other characteristics related to platform implementation, such as processing and distributed scalability. Moreover, we intended to make the cloud computing and big data concepts better in it. Finally, we aim to use the platform to manage a real environment where there are several devices, applications and the possibility of the emergence of new sources of data.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of things for smart cities,” Internet of Things Journal, vol. 1, no. 1, February 2014, pp. 22–32.

[2] T. Nam and T. A. Pardo, “Conceptualizing smart city with dimensions of technology, people, and institutions,” in 12th Annual International Conference on Digital Government Research. ACM, June 2011, pp. 282–291.

[3] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Pichler-Milanovic, and E. Meijers, “Smart cities: Ranking of european medium-sized cities,” Centre of Regional Science (SRF), Tech. Rep., 2007.

[4] A. Mostashari, F. Arnold, M. Maurer, and J. Wade, “Citizens as sensors: The cognitive city paradigm,” in 8th International Conference Expo on Emerging Technologies for a Smarter World (CEWIT). IEEE, November 2011, pp. 1–5.

[5] K. Su, J. Li, and H. Fu, “Smart city and the applications,” in International Conference on Electronics, Communications and Control (ICECC). IEEE, September 2011, pp. 1028–1031.

[6] K. Gama, L. Touseau, and D. Donsez, “Combining heterogeneous service technologies for building an internet of things middleware,” Computer Communications, vol. 35, no. 4, November 2012, pp. 405–417.

[7] G. Cugola and A. Margara, “Processing flows of information: From data stream to complex event processing,” ACM Computing Surveys, vol. 44, no. 3, June 2012, pp. 15:1–15:62.

[8] L. Anthopoulos and P. Fitsilis, “From digital to ubiquitous cities: Defining a common architecture for urban development,” in Sixth International Conference on Intelligent Environments (IE). IEEE, July 2010, pp. 301–306.

[9] L. Filipponi, A. Vitaletti, G. Landi, V. Memeo, G. Laura, and P. Pucci, “Smart city: An event driven architecture for monitoring public spaces with heterogeneous sensors,” in Fourth International Conference on Sensor Technologies and Applications. IEEE, July 2010, pp. 281–286.

[10] M. Blackstock, N. Kaviani, R. Leal, and A. Friday, “Magic broker 2: An open and extensible platform for the internet of things,” in Internet of Things (IOT). IEEE, November 2010, pp. 1–8.

[11] B. Valente and F. Martins, “A middleware framework for the internet of things,” in The Third International Conference on Advances in Future Internet. IARIA, 2011, pp. 139–144.

[12] F. Andreini, F. Crisciani, C. Cicconetti, and R. Mambrini, “A scalable architecture for geo-localized service access in smart cities,” in Future Network and Mobile Summit (FutureNetw). IEEE, June 2011, pp. 1–8.

[13] P. Bakker and B. Ertman, Building Modular Cloud Apps with OSGi. USA: O Reilly, 2013.

[14] R. K. Yin, Case Study Research: Design and Methods. SAGE Publications, 2003.

[15] B. Kitchenham, L. Pickard, and S. L. Pfleeger, “Case studies for method and tool evaluation,” IEEE software, vol. 12, no. 4, July 1995, pp. 52–62.

[16] P. Runeson and M. Host, “Guidelines for conducting and reporting case study research in software engineering,” Empirical software engineering, vol. 14, no. 2, April 2009, pp. 131–164.