Published at the 6th International Conference on Smart Cities and Green ICT Systems, April 22 - 24, 2017

InterSCity: A Scalable Microservice-based Open Source Platform for Smart Cities

Arthur de M. Del Esposte[1], Fabio Kon[1], Fabio M. Costa[2] and Nelson Lago[1]
[1]*Department of Computer Science, University of São Paulo*
[2]*Instituto de Informática, Universidade Federal do Goiás*
{*esposte, kon, lago*}*@ime.usp.br, fmc@inf.ufg.br*

Keywords: Smart Cities, Software Platform, Microservices, Scalability, Open Source Software

Abstract: Smart City technologies emerge as a potential solution to tackle common problems in large urban centers by using city resources efficiently and providing quality services for citizens. Despite the various advances in middleware technologies to support future smart cities, there are no universally accepted platforms yet. Most of the existing solutions do not provide the required flexibility to be shared across cities. Moreover, the extensive use and development of non-open-source software leads to interoperability issues and limits the collaboration among R&D groups. In this paper, we explore the use of a microservices architecture to address key practical challenges in smart city platforms. We present InterSCity, a microservice-based open source smart city platform that aims at supporting collaborative, novel smart city research, development, and deployment initiatives. We discuss how the microservice approach enables a flexible, extensible, and loosely coupled architecture and present experimental results demonstrating the scalability of the proposed platform.

1 INTRODUCTION

The rapid growth of cities around the world has created large, densely populated urban centers characterized by complex interconnected structural, social and economic organizations. This urbanization phenomenon imposes several challenges for sustainable development and quality of life in cities that traditional management approaches cannot overcome. Thus, *smart cities* emerge as a new paradigm aimed at addressing these problems by using city resources efficiently and providing quality services for its citizens. As a consequence, several efforts have been devoted to the study and development of smart city solutions to address the major problems faced by the cities of today such as traffic jams, air pollution, and energy efficiency.

Smart cities are characterized by the adoption of Information and Communication Technologies (ICT) as an integral part of the city's infrastructure to support multiple solutions for urban challenges (Neirotti et al., 2014). The Internet of Things (IoT), Big Data, and Cloud Computing are key enabling technologies of smart cities that offer a wide range of opportunities and challenges, both in the academy and industry. To fully exploit the potential of these enablers, future smart cities will demand a unified ICT infrastructure to properly share their resources rather than relying on non-integrated solutions, or market islands (Villanueva et al., 2013). The most common approach still adopted in existing initiatives is to develop ad-hoc applications for specific domains, such as health care and urban mobility, which leads to low resource sharing and the proliferation of non-interoperable services.

Many authors advocate that integrated middleware platforms can provide the required infrastructure to support the construction of sophisticated, cross-domain smart city applications (Villanueva et al., 2013; Hernández-Muñoz et al., 2011; Fazio et al., 2012). Such platforms must include facilities for application development, for enabling interoperability between the different systems of a city, for managing large amounts of data, and for dealing with a heterogeneous number of distributed devices and services at city scale, to cite only a few requirements (Santana et al., 2016).

Despite the various advances in middleware technologies, protocols, and standards, many aspects related to the design, development, deployment, and

*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.

management of smart city platforms still challenge the research community. Consequently, there are no universally accepted platforms yet, and existing solutions do not provide the required flexibility to be shared across cities (PCAST, 2016). In the following, we highlight three key factors that contribute to the lack of practical and reusable solutions in the field.

First, in addition to traditional functional capabilities, smart city platforms must meet a number of non-functional requirements to enable their use in different environments and applications. Security and privacy policies may change according to the laws and regulations of the city, impacting design decisions regarding storage and availability of the data in the platform. Different contexts may expose a great diversity of requirements, which may dynamically evolve over time. Thus, smart city platforms must provide a flexible architecture to adopt new technologies and support new functional and non-functional requirements to suit the diversity of the multiple and constantly evolving city environments where they are deployed. Similarly, a smart city platform must scale up to handle the increasingly large number of users, devices, and services as well as their associated data, which will grow with the evolution of urban environments. In particular, these platforms must offer different scalability strategies by design to meet the diverse scalability demands. Therefore, the dynamism and continuous evolution of urban environments require the use of new approaches to develop flexible, evolvable, maintainable, and scalable architectures for smart city platforms (Krylovskiy et al., 2015).

The second point is related to the lack of practical and scientific validation to evaluate the different aspects of smart city solutions, such as social and economic impact, internal and external quality, performance, scalability, and feasibility, to cite a few. In (Sanchez et al., 2011), the authors observed that although many IoT projects present concrete solutions, validation of the developed technologies and architectural models are limited to proofs-of-concept, not allowing conclusive results. Therefore, we still need more comprehensive studies with experiments and tests that allow the comparison of different smart city technologies. For this purpose, significant effort should be devoted to (1) deploying existing solutions in real production scenarios, (2) developing methods and tools for more sophisticated simulation-based evaluations, and (3) developing well-defined benchmark strategies for cross-platform evaluation.

Lastly, the extensive use and development of non-open-source software in the core of smart city platforms jeopardize their widespread adoption as it leads to interoperability difficulties and limits the collaboration among R&D groups, often forcing them to "reinvent the wheel". The use of open technologies is crucial to the sustainability and development of future smart cities, since they prevent vendor lock-in, enable collaborative development, market opportunities, and sharing of solutions.

To address the above-mentioned challenges, we propose InterSCity, an open source microservices-based platform for smart cities. Its objective is to provide a high-quality, modular, highly scalable middleware infrastructure to support smart city solutions that can be reused across cities and R&D groups, as well as governments and companies.

Microservice architectures emerged from the software industry's best practices in building large-scale distributed applications composed of small, interconnected components (microservices), supporting scalability, evolvability, maintainability, and modularity. InterSCity leverages the microservice approach to implement the fundamental modules described by the reference architecture proposed in (Santana et al., 2016), conceived from the analysis of 23 smart city projects. This reference architecture describes the building blocks needed to meet the main functional and non-functional requirements to guide the development of next-generation platforms for smart cities. Thus, the platform provides high-level services to manage heterogeneous IoT resources, data storage and processing, and context-aware resource discovery. In this paper, we discuss the design and implementation details of the InterSCity platform to address the design challenges in smart city systems.

This paper brings two key contributions to smart cities platforms research: (I) the InterSCity platform as an open-source project to enable novel smart city research, development, and deployment initiatives; (II) an analysis of the impact of its microservice architecture to address key research challenges related to *scalability* and *evolvability* in smart city platforms based on experimental results and our early experience. Section 2 presents a discussion of related work. Section 3 describes in detail the InterSCity microservice architecture, design principles, and an example application. Section 4 presents a scalability evaluation of the platform within a scenario of data streaming from sensors spread across a city. Lastly, we present the concluding remarks and future work in Section 5.

2 RELATED WORK

Several efforts have been devoted to the study and development of platforms that address the key chal-

lenges of smart cities. In particular, projects that aim at addressing the practical problems related to the development, deployment, and maintenance of smart city services and applications are the most relevant in the context of this work.

Perhaps the most notable project that targets the realistic deployment and validation of smart city solutions is SmartSantander, which provides a smart city testbed with research facilities composed of more than 20,000 IoT devices deployed in urban environments (Sanchez et al., 2014). The SmartSantander testbed aims at supporting experimentation with smart city services in a realistic setting at a large scale. Despite the relevance of this project to enable experiments in real scenarios, it is not clear whether it meets important requirements that would allow its applicability in other contexts (e.g., in cities with different characteristics), such as adaptability, flexibility, and extensibility. We advocate that an open source smart city platform based on a scalable microservices architecture is an effective way to achieve such requirements.

A number of middleware platforms were developed in recent years to address multiple requirements towards the construction of cross-domain smart city solutions, as opposed to traditional approaches based on vertical silos. The Civitas middleware (Villanueva et al., 2013) fulfills the main functional requirements by proposing a set of essential standards and tools to enable smart city ecosystems. The Gambas (Apolinarski et al., 2014) project offers tools to facilitate the development and deployment of smart city applications, including a runtime environment and an SDK. However, it is not clear whether these platforms address some of the key non-functional requirements, such as scalability, extensibility and evolvability.

OpenIoT[1] is one of the most relevant projects handling the main requirements of smart city platforms. It is an open source layered middleware platform that aims at enabling semantic interoperability across IoT applications, including smart cities (Soldatos et al., 2015). The platform provides visual tools to facilitate the administration and implementation of applications directly on top of it. Although OpenIoT provides several facilities to support IoT applications, it is not clear how its architecture deals with important aspects related to scalability, adaptability, and extensibility.

Krylovskiy et al. (Krylovskiy et al., 2015) present the DIMMER project, a microservice-based IoT platform to support applications that aim at improving energy efficiency and management in cities. This is the only previous work we have found that explores the use of a microservices architecture to build smart city platforms. The authors described their early experience in adopting microservices, covering their impact on organizational and design aspects. However, their work leaves several open questions that demand further experimental and empirical studies to obtain more conclusive results on the adoption of microservices in the design of smart city platforms, such as on issues related to scalability and perfomance. Our work has a broader scope, as the InterSCity platform is designed to support smart city applications from multiple domains and is fully developed as an open source project, as opposed to the DIMMER platform.

[1]https://github.com/OpenIotOrg/openiot

3 THE INTERSCITY PLATFORM

InterSCity is a project aiming at developing multidisciplinary, high-quality scientific and technological research to address the key challenges related to the software infrastructure of smart cities (Batista et al., 2016). One of the main goals of InterSCity is to develop reusable open source technologies and methods to support future Smart Cities. In this context, the InterSCity platform is a concrete result from the initial research efforts in the project.

The InterSCity platform has been developed incrementally as an open source microservices-based platform to enable collaborative research, development, and experiments in smart cities. The platform was designed from the outset following the reference architecture for smart city platforms proposed in (Santana et al., 2016). This reference architecture aims at guiding the development of next-generation smart city platforms by describing and organizing the major building-blocks that are required to meet a wide range of functional and non-functional requirements elicited from the analysis of a large number of existing smart city projects. By implementing key building-blocks of this architecture, the InterSCity Platform covers the major features required to support integrated smart city applications in different domains, such as public transportation, public safety, and environmental monitoring. Currently, the InterSCity Platform provides a set of high-level cloud-based services to manage heterogeneous IoT resources, data storage and processing, and context-aware resource discovery.

We follow agile software development methods aligned with practices of open-source project communities to provide a high-quality platform that can be shared and collaboratively developed among cities, research groups and development communities. The

source code is available online [2].

3.1 Design Principles

Smart cities emerge from advances in tools and techniques developed both in the industry and academia, such as the Internet of Things, Big Data, and Cloud Computing. However, the integration of these tools and technologies is not straightforward, as these areas evolve rapidly and their combination raises complex issues. This brings new challenges, approaches, and dynamics that result in the constant emergence of novel technologies, standards, and services. Smart city applications further enhance the dynamism of the involved technologies, since they encompass complex environments composed of several interconnected subsystems which are constantly evolving and presenting new challenges. Such dynamism impacts several design decisions in smart city platform architectures.

To provide a high-quality, practical smart city platform to support future smart city projects, we must address two key design issues that jeopardize the wide adoption of existing solutions in different smart city initiatives: *scalability* and *evolvability*

Scalability - A platform must scale well in multiple dimensions to properly support a smart city. Among others, smart city platforms must handle: (I) a large number of devices that compose the city IoT infrastructure; (II) millions of users and components that use the platform services; (III) a very large volume of city-related data that must be stored and processed; (IV) a potentiality large set of new services that can interact with the platform to offer complementary capabilities. The scalability requirements may vary depending on the context and should be addressed since the beginning of any smart city project.

Evolvability - Urban environments are very dynamic and tend to change constantly in terms of organization, regulations, problems, opportunities, and challenges. Therefore, smart city platforms must be adaptable to meet changing requirements in a cost-effective way. Evolvability is the system's capability to evolve over time by supporting rapid modification and enhancement with low cost and small architectural impact, and is a fundamental element for the success and economic value of long-lived software (Breivold et al., 2012)

Our strategy to address scalability and evolvability issues and to provide the required services to support smart cities is to adopt a microservices architecture for the platform. The microservices model emerged from the software industry efforts to build large-scale distributed systems refining SOA guidelines through the adoption of DevOps and Agile principles, tools, and techniques. Although there are efforts to understand the impact of this architecture on other research areas (Le et al., 2015; Gopu et al., 2016), very few works explore the potential of microservices in the context of smart cities.

Here we present the design principles adopted in the InterSCity Platform which are aligned with microservices patterns, which are critical for the wide adoption of the platform in different smart city projects.

- **Modularity via Services.** Modularity is a key concept used in software architecture to divide systems into smaller functional units. Microservice architectures achieve modularity through single-purpose, small services that communicate through lightweight mechanisms to achieve a common goal.
- **Distributed Models and Data.** In monolithic architectures and even in traditional service-oriented systems, it is fairly common to create a unified domain model and a centralized storage backend. With microservices, each service has its own database and models, which may evolve independently of external services. Decentralized data management and the possibility to use different technologies that best fit each context are relevant advantages. On the other hand, increased operational complexity is the main drawback.
- **Decentralized Evolution.** Microservices must be autonomous, providing well-defined boundaries and communication APIs so that they can evolve and be maintained independently. Moreover, this principle ensures that each service may implement its functionalities using the most appropriate technology, provoking positive *technology heterogeneity*. Similarly, each microservice may *scale independently* using different strategies, since scalability requirements vary across services. Finally, this design principle should reflect on the configuration and deployment procedures, which may be performed independently as well.
- **Reuse of Open Source Projects.** Reusing software components is a fundamental practice of software engineering to achieve productivity, cost effectiveness and software reliability. We always give preference to the use of existing robust open-source tools, libraries, and frameworks instead of implementing components from scratch, since the quality of popular open-source packages is admittedly good as already empirically observed (Taibi,

[2] https://gitlab.com/smart-city-software-platform

2013). Moreover, we adopt a rigorous technology selection criteria and only incorporate open source components that have an active developer community, stable release support, and appropriate documentation to guide usage, development, and deployment.

- **Adoption of Open Standards.** As important as the reuse of open-source projects is the adoption of open, well-accepted standards that are designed to provide interoperability at different levels. This prevents technology and vendor lock-in. The use of open Internet and web standards is essential to enable the *true* Internet of Things, being widely used in related projects found in the literature (Fazio et al., 2012; Amaral et al., 2015; Hernández-Muñoz et al., 2011).
- **Asynchronous versus Synchronous.** Although most services provide RESTful APIs, they must implement asynchronous messaging as much as possible to avoid blocking in synchronous request-reply interactions. Asynchronicity should be achieved by using notifications, the publish/subscribe design pattern, and event-based communication strategies to support low latency and scalability. Besides, the platform must rely on a lightweight message bus with the single purpose of providing a reliable messaging service rather than traditional SOA approaches that use sophisticated, heavy middleware such as an Enterprise Service Bus (ESB).
- **Stateless Services.** This design principle supports scalability by advocating that services should be stateless to enable any service instance to respond to any request, facilitating load distribution and elasticity. Thus, the design of microservices should separate state data, such as context and session data, to be managed by an external component whenever possible.

3.2 Platform Architecture

The InterSCity microservices architecture is shown in Figure 1, addressing important aspects of IoT and Data management, providing high-level RESTful services to support the development of smart city applications, services and tools for different purposes. The underlying IoT Gateways can register new devices to the platform and send sensor data through a REST API. InterSCity abstracts the complexity involved in the communication between smart city applications and IoT devices, as well as the complexity of city-scale data management.

InterSCity provides well-defined boundaries to communicate with both IoT devices and smart city applications. Currently, the platform is composed of six different microservices that provide features for the integration of IoT devices (Resource Adaptor), data and resource management (Resource Catalog, Data Collector and Actuator Controller), resource discovery through context data (Resource Discovery) and visualization (Resource Viewer). Although all microservices expose REST APIs for synchronous messaging over HTTP, most of the communication for the composition of services is done through asynchronous calls, relying on the lightweight message bus RabbitMQ[3] for asynchronous messaging through the Advanced Message Queuing Protocol (AMQP).

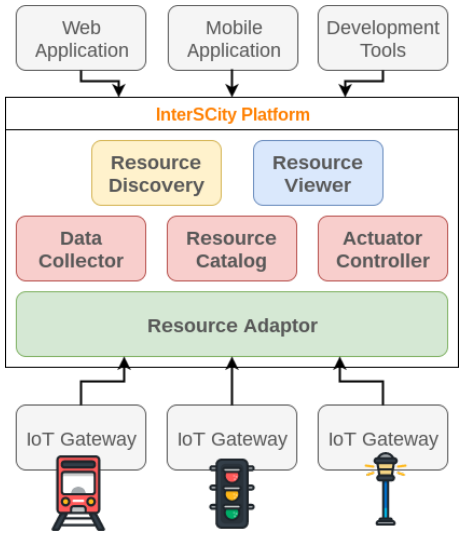

Figure 1: The InterSCity Platform Architecture.

Interactions between the platform and its clients involve the manipulation of *city resources*. A city resource is a logical concept that encapsulates a physical entity that makes up the city, such as cars, buses, traffic lights, and lampposts. Resources comprise attributes (e.g., location and description) and functional capabilities to provide data and receive commands, which are respectively supported by sensors and actuators coupled to the resource. This approach facilitates the interaction of client applications with a real city environment, since it grants an abstraction composed of city concepts rather than the cyber-physical particulars, that comprise the underlying IoT layers of Smart City ecosystems. As a consequence, for instance, two buses registered in the platform are accessed through the same standardized API regardless of their devices and communication technologies. Likewise, two physical sensor or actuator devices with similar purposes are encapsulated as a common capability, abstracting all the specific details related to

[3] www.rabbitmq.com

data representation and deployment of these devices.

The InterSCity architecture supports a decentralized management of city resources dividing functional responsibilities and data persistence across the microservices. The Resource Adaptor microservice works as a stateless proxy that allows external services, such as IoT gateways, to register and update resources on the platform, post sensed data from those resources, and subscribe to events that indicate actuator commands. All city resources are registered in the Resource Catalog microservice which is responsible for providing universally unique identifiers (UUIDs) (Leach et al., 2005) and for asynchronously notifying the event of resource creation to other microservices.

The Data Collector microservice stores sensor data collected by city resources. Sensor data consist of context information or an event linked to a resource capability which is observed at a particular time. Data Collector provides an API to allow access to both current and historical context data of city resources using a rich set o filters that can be accessed in search endpoints. The Actuator Controller microservice, in turn, provides standardized services to intermediate all actuation requests to city resources with actuator capabilities. Moreover, Actuator Controller records the history of actuation requests so that they can be accessed in the future, e.g., for auditing purposes.

Both the Resource Discovery and the Resource Viewer microservices provide more sophisticated services by orchestrating Data Collector and Resource Catalog. Resource Viewer is a web visualization microservice for presenting city resources information graphically based on Resource Catalog and Data Collector back-end services. The purpose of Resource Viewer is to present general and administrative visualizations of city resources, including location, real time context data, and representative charts of historical data. The Resource Discovery microservice provides a context-aware search API that may be used by client applications to discover available city resources. This API provides filters that can be combined to discover resources. For instance, filters can combine information such as location data, interval rules for current context data, and other types of metadata.

Scalability is a key non-functional requirement for smart city platforms. InterSCity was primarily designed to support smart city projects with a large amount of users, data, and services. Its microservices architecture supports scalability at the functional decomposition level in both application and database tiers by splitting the processing responsibilities across several services, dividing the amount of data into decentralized databases, and isolating services through fine-grained interfaces. However, each microservice will be required to scale at different paces depending on both the specificities of the urban and technological context and the continuously increasing demands. Although the loosely coupled architecture allows microservices to scale out independently, different design strategies must be applied to overcome their own traits in order to achieve horizontal scalability. Table 1 summarizes the scalability strategies currently supported by InterSCity microservices, denoted by ✓ and points out new strategies that will be supported in the near future, denoted by **NF**.

A deployment of the InterSCity platform may have several instances of each of its microservices behind *Load Balancers* to handle higher loads transparently for customers. Services that receive asynchronous messaging, such as Data Collector, Actuator Controller, and Resource Adaptor, are designed to support the addition of more background workers to handle highly intensive demands for event-based jobs. Data Collector also uses database caching supported by Redis[4] to provide low-latency readings of the latest data collected by city resources. Resource Discovery caches static resource meta-data provided by the Resource Catalog, since they do not change very often. Initially, we adopted PostgreSQL[5] in all microservices as it is widely adopted in the software industry and it supports georeferenced queries, which are important in the smart city domain. However, we intend to move towards plurality of database systems in a near future to better fit the scope of each microservice and to properly support horizontal scalability so that the database layer does not become a bottleneck.

The loosely coupled message-oriented communication approach favors the continuous development of the platform, enabling the extension of existing features through service composition and the addition of new microservices to meet the constantly evolving smart city requirements. Moreover, decoupled communication interfaces allow us to maintain microservices in separate code repositories enabling: decentralized version and dependency management, independent, faster tests, safe refactoring, evolution of existing features, and the adoption of the most appropriate technologies in each context. The lower boundary provided by Resource Adaptor enables InterSCity to continuously integrate heterogeneous IoT technologies without affecting other services. It can also be used to integrate the existing legacy ICT infrastructure of cities, such as open data initiatives. It is worth noting that InterSCity' upper API isolates client applications from the modification or addition of new

[4] www.redis.io

[5] www.postgresql.org

Table 1: Scalability strategies supported by InterSCity microservices

Microservice	HTTP Load Balancer	Background Workers	Caching	Database Sharding
Resource Adaptor	✓	✓		
Resource Catalog	✓			NF
Data Collector	✓	✓	✓	NF
Actuator Controller	✓	✓		
Resource Discovery	✓		✓	
Resource Viewer	✓		NF	

technologies in the smart city infrastructure.

InterSCity microservices architecture requires DevOps methods to support both an agile development life cycle and consistent, automated, independent deployments. Thus, we encapsulated microservices into individual Docker [6] lightweight containers which can be deployed and maintained independently. Continuous integration tools play an important role in the automated execution of both individual and integration tests, and to ensure that container images are built correctly along the development of microservices. To perform automated, consistent, remote deployments, we adopted the Ansible[7] automation engine, which provides a set of configuration management tools and scripts, facilitating the deployment of the InterSCity platform and associated applications.

3.3 Application Life-cycle

To demonstrate the aforementioned approaches, here we illustrate how applications can be built over the InterSCity services to interact with city resources. For this purpose, we exemplify the use case of the Smart Parking App, an experimental smart city application developed on top of the platform by University of São Paulo students during a graduate course and whose source code is available in the platform distribution. The Smart Parking application aims to help the difficult task of finding available parking spots in a large city, by offering a map with geolocated real-time information of parking spaces. The system is based on both static and sensor data of individual parking spots equipped with embedded sensors to notify the presence of a parked car. As can be seen in Figure 2, the Smart Parking App allows drivers to discover close parking spaces by offering visualization filters related to their availability, prices, and operating hours. One can check the details of a parking spot and view the route from the user's current location, as shown in Figure 3. The hardware part of this example was simulated via a specific software component that mimicked the behavior of physical sensors.

[6] www.docker.com

[7] www.ansible.com

Figure 4 illustrates a message flow that resulted from the use of the Smart Parking application supported by the InterSCity platform hosted in a cloud infrastructure. This example considers a smart parking infrastructure supported by cyber-physical systems to detect the presence of cars in parking spaces based on technologies that are commonly used in smart parking solutions, such as Wireless Sensor Networks (WSN), Light Dependable Resistor (LDR) sensors, Infra-Red (IR) sensors, and magnetic sensors (Bachani et al., 2016). These sensors send data continuously to a remote IoT Gateway via wireless protocols, such as ZigBee or Bluetooth, as illustrated in Step 1. The responsibilities of the IoT Gateway are two-fold: (I) registering each connected parking spot as a city resource with the *"availability"* sensor capability (Step 2) and (II) notifying the platform when a parking space becomes available or unavailable (Step 3). For these purposes, the IoT Gateway must track resource UUIDs provided by InterSCity to be able to send context data through the Resource Adaptor API upon state change events. We highlight that the concepts of city resource and capability abstract the implementation details of the underlying WSN infrastructure.

To use The Smart Parking application, one must define a target location or automatically use his/her current GPS data in addition to setting custom parameters to filter parking spaces according to the desired characteristics (Step 4). As an example, the application may query the platform for all the parking space resources that match the selected parameters within a 500 meters radius of the target location through the Resource Discovery API (Step 5). As a result, the Smart Parking application renders the current state of the returned parking resources on the map, as shown in Figure 2. Users can set the update time interval to get the current state of the returned resources as well as to modify the parameters, which will result in new requests such as the one performed in Step 5. Additional requests may be performed to get detailed information about a specific parking spot from the Resource Catalog API, such as presented in Figure 3, or even to obtain its availability history through the Data Collector microservice (Step 6).

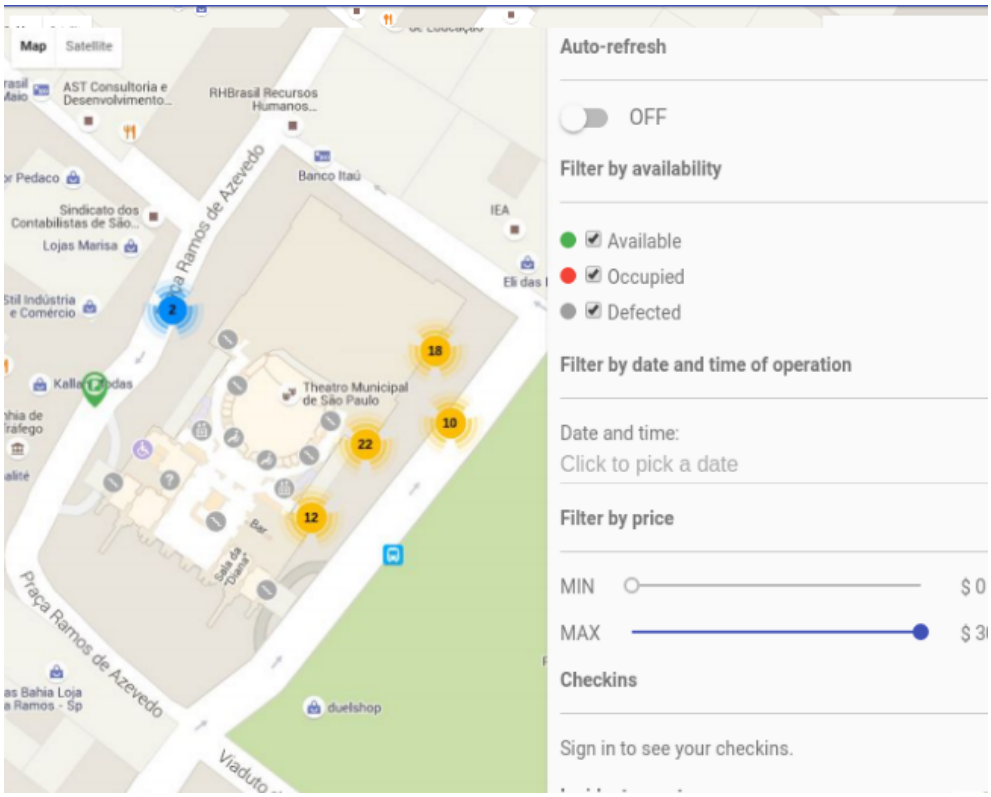

Figure 2: Screenshot of the Smart Parking application.

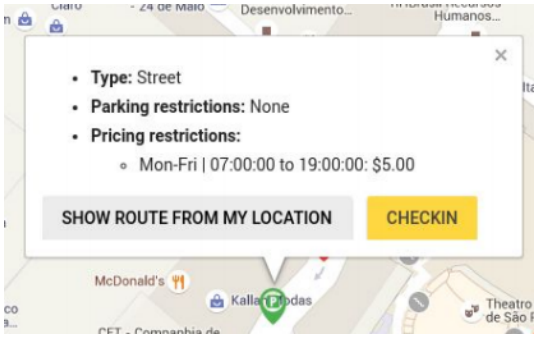

Figure 3: Parking spot details in the Smart Parking application.

4 PERFORMANCE AND SCALABILITY ANALYSIS

To evaluate the proposed platform, we conducted two preliminary experiments. First, we evaluated how the performance of the InterSCity platform degrades with the increase in the number of concurrent IoT gateways (clients) sending sensor data continuously over a long period of time. The main objective of this experiment was to evaluate the individual behavior and consumption of hardware resources of each microservice so that we could identify potential bottlenecks. The second experiment aimed at assessing the scalability of the platform by applying supported scalability strategies to the bottlenecks we identified.

To conduct the experiments, we ran a production-like instance of the InterSCity platform in the Digital Ocean [8] cloud. Both InterSCity microservices and external services, such as PostgreSQL, RabbitMQ, and Redis, were deployed within Docker containers. However, each service instance was hosted on its own virtual machine for isolation purposes, guaranteeing a fixed amount of machine resources per service. In the experiments, we consider a common smart city scenario where distributed sensors continuously collect observations from the city and send the sensed data to IoT gateways. The source code of the scripts used in the experiment are available in the repository for reproducibility [9].

4.1 Degradation Analysis

For the first experiment, we used a total of four single-core GNU/Linux Debian 8.6 machines with 512MB

[8] www.digitalocean.com

[9] https://github.com/LSS-USP/smart-city-platform-experiments

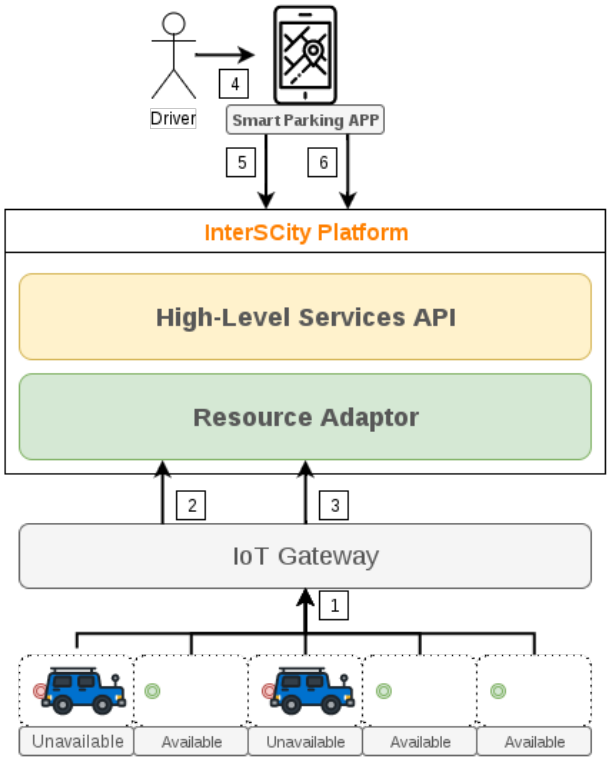

Figure 4: Smart Parking application life cycle.

RAM having 2.0GHz of clock speed in the same private network hosting one single instance of the Resource Adaptor, Resource Catalog, Data Collector, and RabbitMQ services, without any replication. We performed the degradation analysis by benchmarking the platform against 11 different workloads supported by the Funkload[10] load testing tool. Workloads were characterized by the number of concurrent emulated IoT Gateways that continuously send sensor data from city resources to the platform. Each client (the IoT Gateways) runs a loop sending synchronous requests to the platform throughout the experiment as fast as it can. We ran each of the load tests for four minutes, with a 30-second interval between them, collecting both response time and throughput metrics, while keeping the same capacity and configuration of the platform during all workload tests. We repeated each run of the experiment 20 times.

Figure 5 shows the performance degradation of the platform as the number of concurrent IoT Gateways increases. The best average response time occurs for a workload of 50 concurrent clients, which was less than 60 milliseconds.

The average response time remained below 1 second with a workload of up to 350 parallel clients. However, the tests with 250 or more clients in parallel start to have requests with latency above 1 second; for this particular application, this is not a problem, but this is an interesting indication of the limitations of the platform in case of applications with more stringent real-time requirements. It is important to note that the number of failed requests (returned with an error code or with a timeout) varied from 0.01% for 350 concurrent clients to 0.16% for 600 concurrent clients. With up to 250 concurrent IoT Gateways, 100% of the requests were successful.

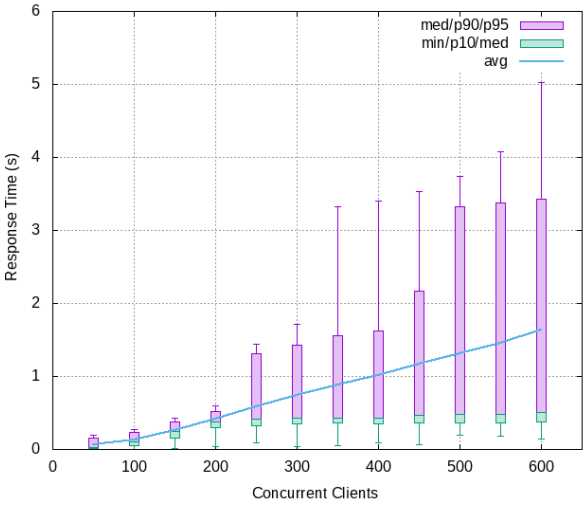

Figure 5: Response time degradation.

In addition to evaluating the degradation in the overall performance, we also monitored the use of hardware resources on each machine to identify potential bottlenecks. For this purpose, we used the Linux-based Collectl [11] tool, as it can be used to monitor a broad set of subsystems such as CPU, disk, memory, processes, and network. Among the four machines used to deploy the platform in the experiment, the one that hosted the Resource Adaptor microservice presented the highest load in CPU and memory usage, both close to 100% during the entire duration of the experiment, being identified as the first bottleneck in the tested scenario. The Data Collector host machine also maintained a high level of CPU usage, as well as an intensive use of I/O operations to store the sensed data. The other two machines did not characterize bottlenecks.

4.2 Scalability Analysis

The objective of the second experiment was to evaluate the scalability of the InterSCity platform, as well

[10] funkload.nuxeo.org

[11] collectl.sourceforge.net

as to demonstrate the flexibility of our architecture to address scalability issues. The same smart city scenario was considered, with concurrent IoT gateways continuously sending sensor data from city resources to the platform. For this experiment we kept a fixed workload with 500 concurrent clients to evaluate the platform's speedup and scale-up metrics. The speedup metric measures how the performance improves with the addition of new resources to the system, while the scale-up metric measures the throughput gain.

The loosely coupled microservices architecture allows us to increase only the resources of the identified bottleneck microservices. We benchmarked the platform with the fixed workload during six 4-minute cycles applying a round-robin load balancing strategy by adding a replica of the Resource Adaptor microservice for each new cycle. The first cycle used exactly the same deployment setup of the first experiment described before. Both the Load Balancer (NGINX[12]) and the new Resource Adaptor instances were deployed on Docker containers hosted by separate single core, GNU/Linux Debian 8.6, 512MB RAM, 2.6GHz machines. These tests were performed using the Apache Benchmark[13] Linux tool.

As a result of the scaling strategy, the average response time decreased from 1725 milliseconds (with 1 instance) to 320 milliseconds (with 6 instances). Figure 6 shows the performance improvement measured in the experiment. We can observe a significant performance gain when scaling horizontally only one of the microservices that make up the platform; the speedup is very close to optimal. Since all messages received by Resource Adaptors are published through the RabbitMQ message service, the use of CPU by this service increases considerably, indicating another possibility for improving the speedup even a little more. RabbitMQ offers horizontal scalability natively and we plan to enhance the platform making use of this feature, further improving its scalability.

As can be seen in Figure 7, the mean throughput of the platform increased substantially by horizontally scaling the Resource Adaptor in the tested scenario. With 6 instances, the platform answered an average of 1546 requests per second, 5.5 times more than when using the configuration with a single Resource Adaptor. Similarly to the speedup metric, the scale-up value increases almost at the same rate at which new instances are added, which is an excellent result.

[12]https://www.nginx.com/

[13]httpd.apache.org/docs/2.4/programs/ab.html

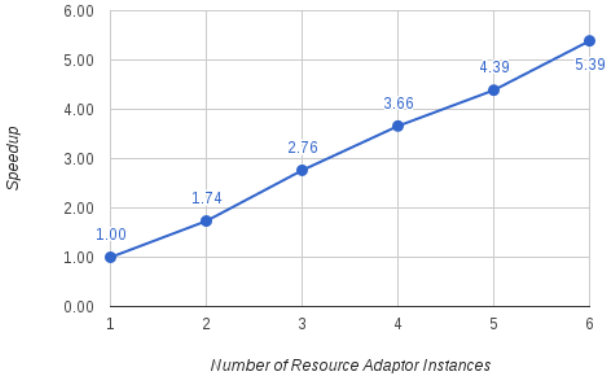

Figure 6: Speedup - performance improvement varying the number of Resource Adaptors.

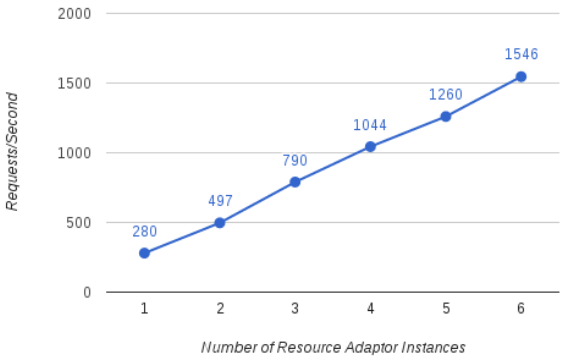

Figure 7: Throughput improvement varying the number of Resource Adaptors.

5 CONCLUSION AND FUTURE WORK

Smart city platforms play a key role in the development of future smart cities as they support cross-domain solutions, interoperability among multiple city systems, and resource and data sharing. However, due to various technical, practical, and methodological challenges, the community still lacks robust solutions that can be shared across smart city initiatives, as well as production environments to support scientific validation of existing proposals. This paper presents two key contributions: (I) advances in the state-of-art by exploring the impact of a microservices architecture in the design, development and deployment of scalable smart city platforms; and (II) a novel microservice-based open source smart city platform that provides facilities to build next-generation scalable smart city solutions and to integrate heterogeneous IoT systems.

Our early experience with the development of InterSCity shows that microservices can be properly used to build smart city solutions that provide finer-

grained, single purpose building blocks that can be more easily and independently evolved compared to traditional SOA approaches. However, it also introduces new challenges due to an increase in the overall complexity. We highlight that the use of industry standards, such as DevOps techniques and open source tools, automated tests, and design patterns, is essential for the successful implementation of our project. In the near future, we intend to investigate more deeply the impact of applying microservices design principles to achieve a loosely coupled, evolvable architecture, demonstrating that InterSCity can be adapted for different smart city environments, can easily integrate new services, and can be modified to meet the constantly evolving city requirements.

Experimental results point towards the applicability of our approach in the context of smart cities, since the platform can support different scalability demands while keeping acceptable performance. These results also show that microservices can be deployed and scaled independently. Further comprehensive experiments should be performed to evaluate all the services provided by the platform. More specifically, we intend to conduct experiments to evaluate the performance and scalability of the InterSCity platform within more realistic scenarios of smart cities, with devices, data, and users at a larger city scale.

Our ongoing work includes several features still needed to meet the constantly evolving requirements of urban environments. This includes more sophisticated Big Data processing and analytics (Al Nuaimi et al., 2015) as well as improved data visualization.

The adopted open source model encourages the community to take advantage of our contribution, as well as to contribute to the InterSCity platform evolution. We expect to enable future smart city initiatives and research from other groups on new approaches to build open, high-quality, practical solutions that can be extended, reused, collaboratively evolved, and deployed in real smart city environments.

ACKNOWLEDGEMENTS

We acknowledge the student members of the Smart Parking App group who develop the application on top of the InterSCity platform: Débora Setton, Hans Harley, Jefferson Silva, Nury Arosquipa, and Thiago Petrone.

We also acknowledge the following developers for their contributions to the InterSCity platform source code: Alander Marques, Ariel Palmeira, Arthur Del Esposte, Athos Ribeiro, Cadu Elmadjian, Caio Salgado, Caroline Satye, Danilo Caetano, Débora Setton, Fernando Freire, Henrique Potter, Igor Lima, João Brito, Leonardo Pereira, Lucas Kanashiro, Macartur Sousa, Marisol Solis, Rodolfo Scotolo, Rodrigo Faria, Rodrigo Siqueira, Rogerio Cardoso, Thiago Petrone, and Wilson Kazuo.

REFERENCES

Al Nuaimi, E., Al Neyadi, H., Mohamed, N., and Al-Jaroodi, J. (2015). Applications of big data to smart cities. *Journal of Internet Services and Applications*, 6(1).

Amaral, L. A., Tiburski, R. T., de Matos, E., and Hessel, F. (2015). Cooperative middleware platform as a service for internet of things applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 488–493, New York, NY, USA. ACM.

Apolinarski, W., Iqbal, U., and Parreira, J. X. (2014). The gambas middleware and sdk for smart city applications. In *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*, pages 117–122.

Bachani, M., Qureshi, U. M., and Shaikh, F. K. (2016). Performance analysis of proximity and light sensors for smart parking. *Procedia Computer Science*, 83:385 – 392. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops.

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

Breivold, H. P., Crnkovic, I., and Larsson, M. (2012). A systematic review of software architecture evolution research. *Information and Software Technology*, 54(1):16 – 40.

Fazio, M., Paone, M., Puliafito, A., and Villari, M. (2012). Heterogeneous sensors become homogeneous things in smart cities. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 775–780.

Gopu, A., Hayashi, S., Young, M. D., Kotulla, R., Henschel, R., and Harbeck, D. (2016). Trident: scalable compute archives: workflows, visualization, and analysis. volume 9913, pages 99131H–99131H–12.

Hernández-Muñoz, J. M., Vercher, J. B., Muñoz, L., Galache, J. A., Presser, M., Gómez, L. A. H., and Pettersson, J. (2011). The future internet. chapter Smart Cities at the Forefront of the Future Internet, pages 447–462. Springer-Verlag, Berlin, Heidelberg.

Krylovskiy, A., Jahn, M., and Patti, E. (2015). Designing a smart city internet of things platform with microservice architecture. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 25–30.

Le, V. D., Neff, M. M., Stewart, R. V., Kelley, R., Fritzinger, E., Dascalu, S. M., and Harris, F. C. (2015). Microservice-based architecture for the nrdc. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1659–1664.

Leach, P. J., Mealling, M., and Salz, R. (2005). A universally unique identifier (uuid) urn namespace. RFC 4122, RFC Editor. `http://www.rfc-editor.org/rfc/rfc4122.txt`.

Neirotti, P., Marco, A. D., Cagliano, A. C., Mangano, G., and Scorrano, F. (2014). Current trends in smart city initiatives: Some stylised facts. *Cities*, 38:25–36.

PCAST (2016). Technology and the future of cities, report to the president. Technical report, Executive Office of the President, United States.

Sanchez, L., Galache, J. A., Gutierrez, V., Hernandez, J. M., Bernat, J., Gluhak, A., and Garcia, T. (2011). Smartsantander: The meeting point between future internet research and experimentation and the smart cities. In *2011 Future Network Mobile Summit*, pages 1–8.

Sanchez, L., Muñoz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., Ramdhany, R., Gluhak, A., Krco, S., Theodoridis, E., and Pfisterer, D. (2014). Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217 – 238. Special issue on Future Internet Testbeds – Part I.

Santana, E. F. Z., Chaves, A. P., Gerosa, M. A., Kon, F., and Milojicic, D. S. (2016). Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *CoRR*, abs/1609.08089.

Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J.-P., Riahi, M., Aberer, K., Jayaraman, P. P., Zaslavsky, A., Žarko, I. P., Skorin-Kapov, L., and Herzog, R. (2015). *OpenIoT: Open Source Internet-of-Things in the Cloud*, pages 13–25. Springer International Publishing, Cham.

Taibi, F. (2013). Reusability of open-source program code: A conceptual model and empirical investigation. *SIGSOFT Softw. Eng. Notes*, 38(4):1–5.

Villanueva, F. J., Santofimia, M. J., Barba, J., and Lópes, J. C. (2013). Civitas: The smart city middleware, from sensors to big data. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 445–450.