Designing a Smart City Internet of Things Platform with Microservice Architecture

Alexandr Krylovskiy*, Marco Jahn*, Edoardo Patti†,
*Fraunhofer FIT, Sankt Augustin, Germany †Dept. of Control and Computer Engineering, Politecnico di Torino, Italy
Emails: {alexandr.krylovskiy, marco.jahn}@fit.fraunhofer.de, edoardo.patti@polito.it

***Abstract*—The Internet of Things (IoT) is being adopted in different application domains and is recognized as one of the key enablers of the Smart City vision. Despite the standardization efforts and wide adoption of Web standards and cloud computing technologies, however, building large-scale Smart City IoT platforms in practice remains challenging. The dynamically changing IoT environment requires these systems to be able to scale and evolve over time adopting new technologies and requirements. In response to the similar challenges in building large-scale distributed applications and platforms on the Web, microservice architecture style has emerged and gained a lot of popularity in the industry in recent years. In this work, we share our early experience of applying the microservice architecture style to design a Smart City IoT platform. Our experience suggests significant benefits provided by this architectural style compared to the more generic Service-Oriented Architecture (SOA) approaches, as well as highlights some of the challenges it introduces.**

I. Introduction

One of the many challenges imposed by the Internet of Things (IoT) is building software systems and platforms that enable support for cross-domain applications, such as Smart City platforms. Many research and standardization efforts has been put into dealing with the heterogeneity of IoT devices and communication protocols [1], [2], as well as service interoperability layers and frameworks [3], [4]. Despite the significant achievements in many of these areas, however, building large-scale IoT systems and platforms capable of evolving and adopting new standards and applications over time remains challenging.

As IoT has become an industry trend supported by large software and hardware vendors like Cisco and IBM, a new wave of start-ups and medium companies offering commercial platforms for building IoT systems emerged [5]. Entering this new and dynamic market, companies adopt the ubiquitous experience of creating large-scale distributed systems on the Web. Fostered by the wide adoption of cloud computing, the contemporary large-scale Web applications and platforms leverage the power of the commodity computing model it provides. The ever changing technology and market require new approaches to build robust architectures that are highly available, scale on demand, and evolve over time. The requirement of decreasing the time to market resulted in the wide adoption of the Agile development practices [6]. The quick release iterations introduced by this adoption created new challenges to operations, which in turn gave birth to the DevOps culture [7] and microservice architecture [8].

Similarly to the way agile software development and DevOps practices have emerged as a reaction to the new challenges in software development and operations, the microservice architecture has appeared from the industry needs to scalability, evolvability, and maintainability of large-scale distributed systems built applying those practices [9]. Microservices have received a wide adoption in the industry among companies building large-scale applications like Amazon and Netflix, as well as Platform-as-a-Service (PaaS) providers like Pivotal [10].

In this work, we share our early experience in applying microservice architectural style to build a Smart City IoT platform for a variety of applications involving different stakeholders to increase the energy efficiency of a city at the district level. We describe the apparent benefits that the microservice architecture provides and new challenges it poses compared to the more traditional Service Oriented Architecture (SOA) approaches for the task of building a service platform for cross-domain applications by an interdisciplinary international team.

The rest of the paper is structured as follows: Section II provides an overview of the related work, Section III introduces the challenges of building Smart City platforms and the microservice architecture. Section IV provides the main contribution describing our experience of designing a Smart City IoT platform applying microservice architecture pattern, and Section V summarizes that experience and future work.

II. Related Work

There are several ongoing research and industry efforts aiming at developing standards and best practices for designing Internet of Things systems and platforms. Due to the great diversity of IoT application domains, there is a high demand in the correspondingly diverse standards capturing the requirements of individual applications at different layers of the protocol stack. In the recent years, a number of standards for the physical, network, and transport layers, as well as security mechanisms tailored to resource-constrained IoT devices have been introduced. The recently standardized CoAP[1] and MQTT[2] protocols together with HTTP finalize the protocol stack by building an application layer. Several industrial alliances and research projects are working on integrating these and new standards in different application domains, enabling interoperability among hardware and software vendors, and defining best practices for building large-scale IoT platforms and applications.

The efforts focusing on interoperability across different application domains include IoT-A [4], OneM2M [3], and FI-WARE [11]. IoT-A is a research project developing an Architectural Reference Model for IoT solutions. It promotes

common understanding of the problem space by providing an IoT Reference Model, describes essential building blocks of an IoT solution by defining a Reference Architecture, and guides architects in designing IoT solutions by providing Guidelines. It does not cover implementation aspects or define new interoperability standards, instead providing a conceptual framework and best practices for designing IoT solutions supporting interoperability.

OneM2M [3] is a global telecom initiative for interoperability of M2M (Machine-to-Machine) and IoT devices and applications. Its main goal is to develop a common specification of a Service Layer Platform that builds on the existing IoT and Web standards, defining specifications of protocols and service APIs. Providing high-level APIs of services, OneM2M defines a specification for interoperability of IoT platforms at the service layer. However, while providing detailed description of functionality, protocols, and APIs of platform services, it leaves open many implementation details. For example, it does not cover the scalability, availability, and deployment aspects of the IoT platforms implementing these services.

FI-WARE [11] is a research project aiming at building a platform for the Future Internet that would provide a novel service infrastructure built of reusable components (Generic Enablers). The vision is that to build an IoT service platform for the application domain at hand, one would select existing Generic Enablers from the FI-WARE catalog and complement them by implementing additional Specific Enablers. FI-WARE is an ongoing project and many of the Generic Enablers constituting the core platform are under development. Several systems for specific use cases have been developed by the partners of the FI-WARE project so far [12], [13], and it remains to be seen whether it will receive adoption in the wider community. In [14], authors show that the level of generalization provided by the FI-WARE platform may lead to overly complex architecture in simple applications.

The experience shared in [14] highlights one of the main problems inherent to standardization efforts like FI-WARE that result in a high level of generalization. The excessive generalization in standards for systems and service platforms poses limitations on both the flexibility of their implementation and their use by applications. The experience of the Web and the success of distributed systems built using its basic principles [15] encourages simple standards and flexibility in their implementation.

Using Web standards and experience is recognized as a common approach to building IoT platforms. E.g., the urban IoT system described in [16] is built using RESTful Web services approach to design the service platform part of the system. Successfully applying this approach to designing an interoperable Smart City platform, authors highlight its benefits in enabling cross-domain applications while reusing the existing development experience of the Web. The Web-based approach is also recommended by the IoT-A and has been successfully adopted in other projects to build Smart City platforms, e.g., SmartSantander [17] and ALMANAC [18].

Building on the previous experience of adopting Web-based approach to designing IoT platforms, in this work we focus on several practical aspects of it following the microservices architectural principle. This includes the high-level platform design and its componentization into services, interaction of services with each other and consuming applications, as well as the platform deployment and operational aspects. In addition to that, we discuss how services using IoT-specific protocols can be integrated in the service platform preserving its design and operational principles.

III. Designing Smart City IoT Platform

A. Challenges

The Smart City vision is to make a better use of the public resources, increasing the quality of the services offered to the citizens while reducing the operational costs of the public administrations [16]. Realizing this vision involves building a large-scale urban IoT system and a service platform on top of it that would provide access to the IoT data and Smart City services. The latter is represented by a large variety of services with varying requirements to the platform infrastructure [16]. Considering the early stage of the IoT development and its progressive adoption, the Smart City IoT platforms designed today need to be able to support new standards and services in the future.

Designing large-scale distributed systems that evolve as the underlying technology and requirements change is one of the challenges addressed by the modern Web and cloud applications. The simplicity of interfaces and loose coupling of individual components promoted by the REST architectural principles coupled with the commodity computing model and elasticity provided by the cloud build the foundation of modern distributed Web applications.

The success of commercial platforms in the growing IoT field [5] shows the successful adoption of the architectural principles of the Web and the experience of building large-scale distributed applications in the cloud. The apparent benefits of the IoT and cloud integration [19] motivates the adoption of the broader cloud experience in building and provisioning large-scale distributed systems that are designed to scale and evolve over time for building IoT platforms.

B. Microservice Architecture

One of the recent trends in the practices of building distributed Web applications is microservice architecture. Emerged as a pattern from the real-world experience of building distributed applications, it does not have a formal definition. Informally, it can be defined as *an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms [8]*. These services are *small, highly decoupled and focus on doing a small task* [20].

Services are the building blocks comprising the systems built with microservice architecture. They define the main characteristics and competitive advantages of these systems, as well as differentiate this architectural approach from others falling under the Service Oriented Architecture (SOA) umbrella. The key characteristics of the microservice architecture relevant in the context of this work are described below.

Componentization via Services. Componentization or modularity are considered as a generally good practice in software engineering, yet achieving it often deems challenging.

With microservice architecture, componentization is achieved via breaking systems down into services, which are independently replaceable, upgradeable, and deployable. Instead of using in-memory function calls, components in microservice architecture are interacting via service interfaces, which puts restrictions on introducing undesirable tight coupling between components and leaking of functionality from one component into another.

Organization around Business Capabilities. Organization is known to have a significant impact on the systems design [21], and organizations employing microservice architecture tend to practice similar organization of technical teams. More specifically, microservice architecture motivates organization around business capabilities instead of the traditional way of building teams based on the technology layers. This results in cross-functional teams, where each team has the full range of skills required for a specific business area and prevents the "logic everywhere" siloed architectures [8].

Smart endpoints and dump pipes. Microservices commonly use lightweight communication protocols to exchange messages with services keeping their domain logic internal. Compared to the Enterprise Service Bus (ESB) and similar approaches where the communication mechanism provides sophisticated functionality for message transformation and choreography, microservices use the communication medium to barely exchange messages. Whether it is HTTP request-response or a lightweight message bus for asynchronous communication with routing, the business logic in microservice architecture always remains in the endpoints – the services.

Decentralized Governance. Because microservice architecture relies on independently deployable components, the centralized governance of standards and technology platforms can be relaxed. Each service in a system built with microservice architecture can use its own technology that is most suitable for the job. This flexibility in the choice of implementation technology provides the benefits of choosing the best tools and platforms considering their trade-offs, as well as allows to gradually adopt new technologies.

Decentralized Data Management. The microservices architecture enables decentralized data management, implying decentralization in both the conceptual models and the storage backends used by services. The decentralization in conceptual models means that different components (services) have different conceptual models of the world, e.g., by operating with different attributes of the same entities. The decentralization in the storage backend means that every service has its own, independent, storage subsystem that is isolated from other services.

Evolutionary Design. Related to several characteristics described above, evolutionary design is a typical characteristic of microservice architecture where services decomposition is used as a driving force to enable frequent and controlled changes in the system. On the one hand, limited functionality of single components emerging from their focus on small tasks limits the efforts required to introduce changes in individual services. On the other hand, independently deployable and replaceable components together with decentralized governance allow the services to be re-implemented from scratch, possibly using another technology, without affecting the rest of the system.

Exhibiting the described characteristics, systems built with microservice architecture are typically associated with the following benefits [20]:

- **Technology heterogeneity**, also known as polyglot programming and persistence [22], is enabled by the decentralized governance and data management that allows coexistence of different technologies used by different components in the system.
- **Resilience** and ease of deployment are enabled by decomposition via services that provides components with clear boundaries, allowing to isolate failures and gradually degrade the system functionality, as well as update and deploy individual services independently.
- **Scaling** with microservices can be achieved in all of the three axis of the *scaling cube* [23]. In addition to the typical scaling by horizontal duplication (X-axis) and data partitioning (Z-axis), microservices also enable scaling by functional decomposition (Y-axis).
- **Organizational alignment** is enabled by organization around business capabilities and motivates smaller focused teams working on components with smaller code-bases.
- **Composability** follows from the fine-grained componentization via services, enabling creating new system capabilities by composing and re-using existing services. As HTTP protocol and REST APIs are commonly used for communication, microservices also encourage the serendipitous reuse [24].

Despite the described benefits of the microservice architecture, successfully implementing it in practice might be challenging [25]. Decomposing distributed systems into independent granular components, microservices bring the complexity of distributed systems and a great deal of operational overhead. The growing popularity of DevOps culture and the infrastructure automation tools, as well as the accumulated experience in dealing with problems of distributed systems address many of these challenges. Therefore, when designing software systems with microservices, deployment and operational aspects of the resulting systems need to be considered carefully.

IV. DIMMER Platform

The goal of the DIMMER Smart City project [26] is to build a service platform (further referred to as the DIMMER platform) and a number of applications aiming at involving different stakeholders to increase the energy efficiency of a city at the district level. In this section, we describe the current status of the platform design progress highlighting the identified issues and how we address them using the microservice architecture.

The principle architecture of the DIMMER platform is shown in Figure 1. At the conceptual level, it includes heterogeneous Sensor Technologies and District Information Models integrated in the Service Platform that is used by Smart City Applications. The Service Platform encompasses IoT services gathering and processing data from heterogeneous

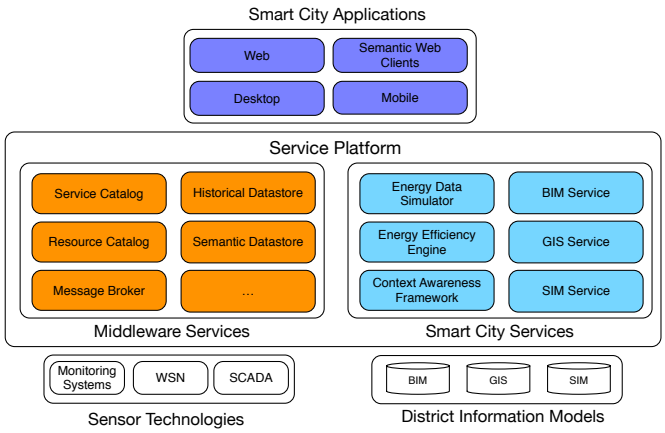

Fig. 1. The principal architecture of the DIMMER platform.

sensor systems (Middleware Services), as well as a number of platform services for Smart City applications (Smart City Services).

Applications using the platform represent a large variety of Web, desktop, and mobile applications for city administration, energy professionals and citizens. In addition to that, the platform can be used by third-party systems and applications, including Semantic Web Clients, to access the data from integrated sensor systems and information models. Such heterogeneity of applications covering diverse use cases and interaction patterns results in a large number of requirements to the platform and its services. In the next sections, we describe these services and their design considerations addressing several specific requirements in more detail.

A. Middleware Services

Middleware is one of the key elements of the IoT platforms as it integrates heterogeneous IoT devices and ICT systems in the platform. The tasks of the middleware include the following: (i) providing modeling abstractions of real-world IoT devices and sensor systems, (ii) enabling search and discovery of these devices and their resources by applications and services, (iii) providing unified APIs and protocols for historical and (near) real-time sensor data access. The DIMMER middleware is based on the LinkSmart OpenSource Middleware [27], extended and complemented by additional components tailored to the requirements of the DIMMER platform.

One of such requirements from DIMMER applications is the support of search and discovery of IoT devices through both lightweight queries over HTTP by mobile applications and SPARQL queries by Semantic Web clients. This simple requirement often cannot be fulfilled due to the internal management of device meta-data in the middleware. Striving to achieve a high level of interoperability and generalization, many middleware systems use Semantic Web to describe and store the device meta-data providing a SPARQL endpoint to query it by the clients [28], [29]. While this might not only be impractical for mobile clients due to the bandwidth limitations, SPARQL also has known limitations in scalability and availability [30].

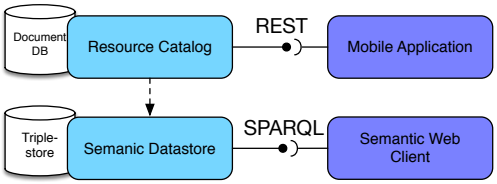

Fig. 2. Accessing IoT meta-data by different clients.

In the DIMMER platform, we use the decentralized data management approach of microservices to manage the IoT devices meta-data in the system. Services in the DIMMER platform use their own views or concepts of the IoT devices (conceptual decentralization) as well as storage backends to store their meta-data (decentralized data storage), building a hierarchical abstraction model. The *Resource Catalog* service of the middleware provides basic information about the devices configuration, deployment, and supported communication protocols. The *Semantic Datastore* service provides a higher-level abstraction, describing the same devices with additional attributes and relations to other entities and platform services using the Semantic Web technologies. Both microservices maintain their *bounded contexts* and are used by different consumers as shown in Figure 2.

In addition to that, middleware provides *Historical Datastore* service for storage of sensor data, *Message Broker* for Publish/Subscribe communication [31] of sensor data, and *Service Catalog* for service discovery. These services use different, domain-specific storage back-ends: *Historical Datastore* service uses timeseries database as it allows to efficiently store and query large amounts of timeseries data (sensor measurements), *Service Catalog* uses a document-based storage as it manages service registrations represented by JSON documents. They also use different protocols: *Message Broker* uses Message Queue Telemetry Transport (MQTT) – the de-facto standard for publish/subscribe communication protocol for the IoT, whereas other services rely on HTTP for request/response communication. As independently deployable microservices, all of them can be scaled and updated separately depending on the particular deployment of the DIMMER platform and its use.

Another important aspect of managing operational (measurements) and meta-data of IoT devices and systems in the middleware is to provide a holistic view of it to the applications and services. Managing meta-data together with the sensor measurements brings the convenience of ubiquitous access to it by consumers via a unified interface that can be implemented by a single service. However, considering the vast amount of data generated by the IoT devices, the resulting network traffic and requirements to the storage and processing infrastructure, storing operational and meta-data together comes at a great cost. Managing them separately, on the other hand, allows to use the infrastructure more efficiently by employing appropriate technologies for these tasks, but introduces complexities to the consumers. Applications and services accessing the IoT data stored separately need to perform several queries to different services, which leads to a more complex client implementation and *chatty* communication.

Using the Application Gateway [20] pattern of microser-

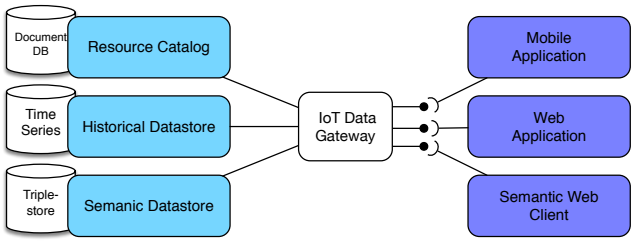

Fig. 3. API Gateway for annotated IoT operational data.

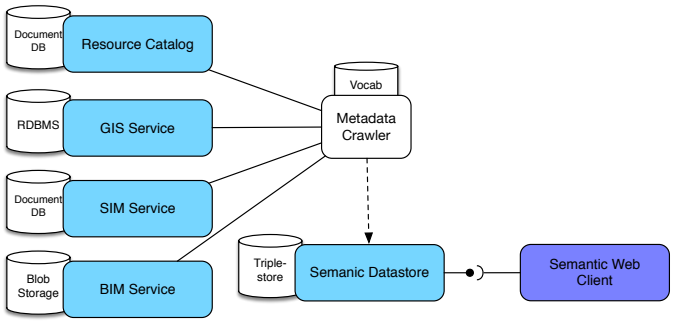

Fig. 4. Semantic Web interoperability for District Information Models.

vice architecture, we can implement an *IoT Data Gateway* service that provides convenient high-level APIs for different kinds of consumers as shown in Figure 3. Built on top of the fine-granular APIs of the underlying microservices, the gateway provides a convenient interface to the IoT data requested by the clients by fanning out client requests to the platform services and returning them in the most convenient form to the corresponding consumers. Depending on the requirements of different consumers and the load generated by them, the API gateway functionality can be implemented in a single service using, e.g., HTTP content negotiation to differentiate between the consumers, or as a set of independent services. The latter would provide more flexible scalability as additional gateways can be deployed to serve only specific kinds of consumers (*Z-axis* scaling).

B. Smart City Services

Smart City services build on top of the IoT middleware and implement the core functionality of the platform. The Smart City services of DIMMER platform include, among others, the following:

- **Services exposing District Information Models** are managing various information models that can be used by Smart City applications. These domain-specific models include data models of Geographic Information Systems (GIS), Building Information Models (BIM), and System Information Models (SIM).
- **Energy Efficiency Engine** is a service providing optimization algorithms and strategies that can be configured and called by the applications using the platform, e.g., applications for analysis and optimization of the district energy consumption at different levels.
- **Energy Data Simulator** is a sensor data simulator used by the applications and other services of the DIMMER platform, e.g., by the Energy Efficiency Engine.
- **Context Awareness Framework** is a set of services providing context awareness features to the applications. Using the Context Awareness Framework, applications can model real-world situations by defining a set of properties as *contexts*, and get notified when those contexts are matched or their states change.

The services exposing District Information Models are easily modeled as separate components, as they have naturally defined context boundaries: every service independently manages its own set of models, using a storage back-end suitable for their data format. Linking different models together, e.g., referencing BIM models of individual buildings in the GIS model, is more challenging, as this is not typically covered by the domain-specific standards. To achieve such interoperability, one could consider defining new standards and data formats, e.g., by using Semantic Web technologies. Professional GIS and BIM tools, however, are working with the well-established data formats in the corresponding domains, and the users of the DIMMER platform working with such tools expect it to export such models in their *native* formats. Mobile and web applications, on the other hand, may use simplified versions of these models for visualization, which requires their transformation in the appropriate data formats.

Providing for such diversity of use cases poses significant challenges on the architecture of the individual services and the whole platform, especially considering that the requirements may change in the future and new use cases appear. Aiming at the evolutionary design, it is important to keep the functionality of individual components within their boundaries while keeping the overall system loosely coupled. In the case of managing District Information Models in the platform, we keep the functionality of individual services managing domain-specific models limited to basic operations on those models, while leveraging Semantic Web and Linked Data technologies for linking different data models to achieve higher-level interoperability.

The interoperability layer is transparent to the domain-specific services and can be implemented as shown in Figure 4. The *Metadata Crawler* is a Linked Data-enabled tool that works similarly to the web crawlers of search engines: it queries available services, annotates the results of these queries using a predefined vocabulary, and exports the annotated data in the *Semantic Datastore*. The latter can be then used by Semantic Web clients to discover the relations between entities represented by domain-specific models and managed by individual services. In this way, the domain-specific services are not concerned with the interoperability layer and can retain their simple implementation, while new interoperability functionalities can be added by modifying the crawler and its vocabulary. This comes at the price of eventual consistency as changes are not immediately propagated throughout the system, but we consider this as a reasonable trade-off in this scenario.

V. Conclusion and future work

While the platform development is at its early stages, our early experience highlights the benefits of using microservice architecture in several aspects, as well as demonstrates some of the challenges that need to be addressed in the future.

First of all, due to the interdisciplinary (software engineers, electrical engineers, architects, energy experts) and international (11 partners from 4 countries) team, using the microservices approach to organization around business capabilities allows us to work highly independently. Together with the decentralized governance and data management, this allows each partner to concentrate on their work while maintaining the overall system compatibility. Agreeing on the service interfaces, every partner is free to choose any technology to implement the platform services of their expertise and work independently from the others. Furthermore, the lack of complex middleware technologies and use of simple communication protocols and APIs instead significantly reduces the amount of coordination work involved.

Standardization of the platform deployment and provisioning infrastructure imposed by the lack of centralized governance and independent implementation technologies provides additional long-term benefits that are apparent even at the early implementation stage. As the DIMMER Platform needs to be deployed in multiple pilot cities, there is a clear need of the standardization on the deployment strategies. With the ubiquity of OpenSource DevOps tools available due to the popularity of microservice architectures in industry, there is a large set of available technologies and tools for us to choose from Moreover, as these tools are designed to work with popular cloud infrastructures, by using them DIMMER Platform can be easily adopted and deployed by teams outside of the project consortium.

Using microservices simplifies the design and implementation of individual services, but comes at the cost of increased complexity of distributed systems. At the current stage, we consider eventual consistency and other compromises made due to the fine-granular service decomposition to be reasonable trade-offs for the gained benefits. However, as more platform services and applications will be developed, a more thorough evaluation needs to be carried out to draw more objective conclusions.

Acknowledgment

This research is funded by EU FP7 SMARTCITIES 2013 District Information Modelling and Management for Energy Reduction – DIMMER.

References

[1] Z. Shelby, K. Hartke, and C. Bormann. (2014) The Constrained Application Protocol (CoAP). [Online]. Available: http://coap.technology/

[2] Message Queue Telemetry Transport (MQTT). [Online]. Available: http://mqtt.org

[3] OneM2M Alliance. [Online]. Available: "http://onem2m.org"

[4] A. Nettstraeter, "Architectural reference model for IoT," *EC FP7 IoT-A (257521) D*, vol. 1, p. 2, 2012.

[5] Postcapes. Internet of Things Platform. [Online]. Available: "http://postscapes.com/internet-of-things-platforms"

[6] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003.

[7] M. Huettermann, *DevOps for developers*. Apress, 2012.

[8] M. Fowler and J. Lewis. (2014) Microservices. [Online]. Available: "http://martinfowler.com/articles/microservices.html"

[9] J. Turnbull, *"The Docker Book: Containerization is the new virtualization"*. James Turnbul, 2015.

[10] M. S. Fred Melo. (2014) Developing Microservices for PaaS with Spring and Cloud Foundry. [Online]. Available: "http://www.infoq.com/presentations/microservices-pass-spring-cloud-foundry"

[11] FI-WARE project. [Online]. Available: "http://fi-ware.org"

[12] D. Havlik, J. Soriano, C. Granell, S. E. Middleton, H. van der Schaaf, A. Berre, and J. Pielorz, "Future Internet enablers for VGI applications," 2013.

[13] T. Usländer, A. J. Berre, C. Granell, D. Havlik, J. Lorenzo, Z. Sabeur, and S. Modafferi, "The future internet enablement of the environment information space," in *Environmental Software Systems. Fostering Information Sharing*. Springer, 2013, pp. 109–120.

[14] D. Namiot and M. Sneps-Sneppe, "On software standards for smart cities: API or DPI," in *ITU Kaleidoscope Academic Conference, Proceedings of the 2014*, June 2014, pp. 169–174.

[15] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[16] A. Zanella, N. Bui, A. P. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, 2014.

[17] E. Theodoridis, G. Mylonas, and I. Chatzigiannakis, "Developing an iot smart city framework," in *Information, Intelligence, Systems and Applications (IISA), 2013 Fourth International Conference on*, July 2013, pp. 1–6.

[18] ALMANAC project Web site. [Online]. Available: "www.almanac-project.eu/"

[19] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "On the integration of cloud computing and internet of things," in *Proceedings of the 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014)*, 2014, pp. 27–29.

[20] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 2015.

[21] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[22] M. Fowler. (2011) PolyglotPersistence. [Online]. Available: "http://martinfowler.com/bliki/PolyglotPersistence.html"

[23] M. L. Abbott and M. T. Fisher, *The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise*. Pearson Education, 2009.

[24] S. Vinoski, "Serendipitous reuse," *IEEE Internet Computing*, vol. 12, no. 1, pp. 84–87, Jan. 2008.

[25] B. Wootton. (2014) Microservices - Not A Free Lunch! [Online]. Available: "http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html"

[26] DIMMER Project. [Online]. Available: "http://dimmer.polito.it"

[27] LinkSmart Middleware. [Online]. Available: "http://linksmart.eu/"

[28] J. Kim and J.-W. Lee, "OpenIoT: An open service framework for the Internet of Things," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 89–93.

[29] P. Kostelnik, M. Sarnovsk, and K. Furdik, "The semantic middleware for networked embedded systems applied in the internet of things and services domain," *Scalable Computing: Practice and Experience*, vol. 12, no. 3, 2011.

[30] D. Rogers. (2013). [Online]. Available: "https://daverog.wordpress.com/2013/06/04/the-enduring-myth-of-the-sparql-endpoint/"

[31] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003.