Soujanya.M* et al.
(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH
Volume No.4, Issue No.1, December - January 2016, 2735 – 2740.

A Study on Smart City IoT Platform with Micro Service Architecture

SOUJANYA. M
M.Tech - Student
Computer Science and Engineering,
R V College of Engineering,
Bangalore, Karnataka, India

Abstract: **A smart city uses information and communication technologies (ICT) to enhance quality, performance and interactivity of urban services, to reduce costs and resource consumption and to improve contact between citizens and government. Sectors that have been developing smart city technology include government services, transport and traffic management, energy, health care, water and waste. Smart city applications are developed with the goal of improving the management of urban flows and allowing for real time responses to challenges. A smart city may therefore be more prepared to respond to challenges than one with a simple 'transactional' relationship with its citizens.**

Key words: **Cyberville, Digital City, Electronic Communities, Flexicity**

I. INTRODUCTION

Major technological, economic and environmental changes have generated interest in smart cities, including climate change, economic restructuring, and the move to online retail and entertainment, ageing populations, and pressures on public finances. The European Union (EU) has devoted constant efforts to devising a strategy for achieving 'smart' urban growth for its metropolitan city-regions. The EU has developed a range of programmes under 'Europe's Digital Agenda". In 2010, it highlighted its focus on strengthening innovation and investment in ICT services for the purpose of improving public services and quality of life. Arup estimates that the global market for smart urban services will be $400 billion per annum by 2020.Examples of Smart City technologies and programs have been implemented in Milton Keynes, Southampton, Amsterdam, Barcelona and Stockholm. It has been suggested that a smart city (also community, Business cluster, urban agglomeration or region) use information technologies to: 1. Make more efficient use of physical infrastructure (roads, built environment and other physical assets) through artificial intelligence and data analytics to support a strong and healthy economic, social, cultural development. 2. Engage effectively with local people in local governance and decision by use of open innovation processes and e-participation, improving the collective intelligence of the city's institutions through E-Governance, with emphasis placed on citizen participation and co-design. 3. Learn, adapt and innovate and thereby respond more effectively and promptly to changing circumstances by improving the intelligence of the city. They evolve towards a strong integration of all dimensions of human intelligence, collective intelligence, and also artificial intelligence within the city. The intelligence of cities "resides in the increasingly effective combination of digital telecommunication networks (the nerves), ubiquitously embedded intelligence (the brains), sensors and tags (the sensory organs), and software (the knowledge and cognitive competence)". This is implemented through: A common IP infrastructure that is open to researchers to develop applications.; Wireless meters and devices transmit information at the point in time.; A number of homes being provided with smart energy meters to become aware of energy consumption and reduce energy usage; Solar power garbage compactors, car recharging stations and energy saving lamps.

II. RELATED RESEARCH WORK

Several ongoing research and industry efforts are 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 Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit

microcontrollers with small amounts of ROM and RAM, while constrained networks such as IPv6 over Low-Power Wireless Personal Area Networks(6LoWPANs) often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation. CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirement such as multicast support, very low overhead, and simplicity for constrained environments. MQTT stands for MQ Telemetry Transport. It is publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium. This is a product from IBM which implements the MQTT protocol in a very scalable manner and which interoperates directly with the Web Sphere MQ family of products. There are other implementations of MQTT listed on the Software page.

OneM2M and different application domains

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. The purpose and goal of oneM2M is to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide.

Technical Specifications and Technical Reports

- Use cases and requirements for a common set of Service Layer capabilities;
- Service Layer aspects with high level and detailed service architecture, in light of an access independent view of end-to-end services;
- Protocols/APIs/standard objects based on this architecture (open interfaces & protocols);
- Security and privacy aspects (authentication, encryption, integrity verification);
- Reachability and discovery of applications;
- Interoperability, including test and conformance specifications;
- Collection of data for charging records (to be used for billing and statistical purposes);
- Identification and naming of devices and applications;
- Information models and data management (including store and subscribe/notify functionality);
- Management aspects (including remote management of entities); and
- Common use cases, terminal/module aspects, including Service Layer interfaces/APIs between:
    - Application and Service Layers;
    - Service Layer and communication functions

FI-WARE – ongoing project

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 FIWARE platform provides a rather simple yet powerful set of APIs (Application Programming Interfaces) that ease the development of Smart Applications in multiple vertical sectors. The specifications of these APIs public and royalty-free. Besides, an open source reference implementation of each of the FIWARE components is publicly available so that multiple FIWARE providers can emerge faster in the market with a low-cost proposition. FIWARE Lab is a non-commercial sandbox environment where innovation and experimentation based on FIWARE technologies take place. Entrepreneurs and individuals can test the technology as well as their applications on FIWARE Lab, exploiting Open Data published by cities and other organizations. FIWARE Lab is deployed over a geographically distributed network of federated nodes leveraging on a wide range of experimental infrastructures. The FIWARE Acceleration Programme aims at promoting the take up of FIWARE technologies among solution integrators and application developers, with special focus on SMEs and start-ups. Linked to this program, the EU launched an ambitious campaign in September 2014 mobilizing 80M€ to support SMEs and entrepreneurs who will develop innovative applications based on FIWARE. Similar programmes may be defined in other regions. Although it was born in Europe, FIWARE has been designed with a global ambition, so that benefits can spread to other regions. The FIWARE Mundus programme is designed to bring coverage to this effort engaging local ICT players and domain stakeholders, and eventually liaising with local governments in different parts of the world, including Latin American, Africa and Asia. Think globally but act locally is a distinguishing mark of the FIWARE ecosystem. The network of FIWARE iHubs will play a fundamental role in building the community of adopters as well as contributors at local level. The FIWARE iHubs Programme aims at supporting the creation and the operations of iHubs nodes worldwide. The experience shared in [14] highlights one of the main problems inherent to standardization efforts similar to FIWARE that result in a high level of generalization. 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., SmartSantander1 and ALMANAC2.

Microservice Architecture

One of the recent trends in the practices of building distributed Web applications is micro service 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 [18]. Although our natural inclination is to pass such things by with a contemptuous glance, this bit of terminology describes a style of software systems that we are finding more and more appealing. We've seen many projects use this style in the last few years, and results so far have been positive, so much so that for many of our colleagues this is becoming the default style for building enterprise applications. Sadly, however, there's not much information that outlines what the micro service style is and how to do it. In short, the micro service architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. To start explaining the micro service style it's useful to compare it to the monolithic style: a monolithic application built as a single unit. Enterprise Applications are often built in three main parts: a client-side user interface (consisting of HTML pages and JavaScript running in a browser on the user's machine) a database (consisting of many tables inserted into a common, and usually relational, database management system), and a server-side application. The server-side application will handle HTTP requests, execute domain logic, retrieve and update data from the database, and select and populate HTML views to be sent to the browser. This server-side application is a monolith - a single logical executable. Any changes to the system involve building and deploying a new version of the server-side application. Such a monolithic server is a natural way to approach building such a system. All your logic for handling a request runs in a single process, allowing you to use the basic features of your language to divide up the

application into classes, functions, and namespaces. With some care, you can run and test the application on a developer's laptop, and use a deployment pipeline to ensure that changes are properly tested and deployed into production. You can horizontally scale the monolith by running many instances behind a load-balancer. Services are the building blocks comprising the systems built with micro service 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 micro service 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 micro service 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 micro service 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 [19], and organizations employing micro service architecture tend to practice similar organization of technical teams. More specifically, micro service 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

Micro services commonly used 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, micro services 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 micro service architecture always remains in the endpoints – the services.

Decentralized Governance

Because micro service architecture relies on independently deployable components, the centralized governance of standards and technology platforms can be relaxed. Each service in a system built with micro service 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

Micro service architecture enables decentralized data management, implying decentralization in both the conceptual models and the storage back ends 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.

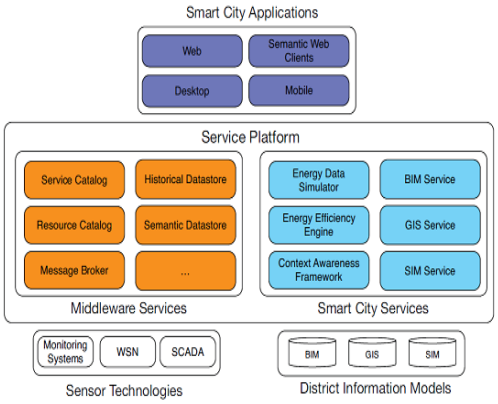
Evolutionary Design

Related to several characteristics described above, evolutionary design is a typical characteristic of micro service 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 micro service architecture are typically associated with the following benefits [18]:

Technology Heterogeneity

It is also known as polyglot programming and persistence [20], 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 micro services can be achieved in all of the three axis of the *scaling cube* [21]. In addition to the typical scaling by horizontal

duplication (X-axis) and data partitioning (Z axis), micro services 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, micro services also encourage the serendipitous reuse [22]. Despite the described benefits of the micro service architecture, successfully implementing it in practice might be challenging [23]. Decomposing distributed systems into independent granular components, micro services 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 micro services, deployment and operational aspects of the resulting systems need to be considered carefully.

III. CONCLUSIONS

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 administration. 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 .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.

IV. ACKNOWLEDGEMENTS

The author thanks Dr. S. Sridhar, Professor and Director, RV Cognitive & Central Computing, R. V. College of Engineering, Bangalore, India for communicating this article to this Journal for publication. The author also thanks Dr. G. Shobha, Professor and Head, Department of CSE, R. V. College of Engineering, Bangalore, India for the support given.

V. REFERENCES

[1]. Alexandr Krylovskiy, Marco Jahn, Edoardo Patti, Fraunhofer FIT, Sankt Augustin, Germany Dept. of Control and Computer Engineering, Politecnico di Torino, Italy "Designing a Smart City Internet of Things Platform with Microservice Architecture" IEEE Internet of Things Journal, 2015

[2]. The Constrained Application Protocol (CoAP), IEEE Std. rfc7252, 2014.

[3]. Message Queue Telemetry Transport (MQTT), OASIS Std., Rev. 3.1.1, 2014.

[4]. OneM2M Alliance. (2014) Onem2m: Standards for m2m and the internet of things. [Online]. Available: http://onem2m.org

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

[6]. Postcapes. (2014) Internet of Things Platforms. [Online]. Available: "http://postscapes.com/internet-of-things-platforms"

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

[8]. M. Huettermann, DevOps for developers. Apress, 2012.

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

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

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

[12]. FI-WARE. (2014) FI-WARE project. [Online]. Available: ”http://fiware.org”

[13]. D. Havlik, J. Soriano, C. Granell, S. E. Middleton, H. van der Schaaf, Berre, and J. Pielorz. (2013) Future Internet enablers for VGI applications. [Online]. Available: http://eprints.soton.ac.uk/370583/

[14]. T. Usl¨ander, 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.

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

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

[17]. Botta, W. de Donato, V. Persico, and A. Pescap´e, “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.

[18]. S. Newman, Building Microservices. O’Reilly Media, Inc., 2015.

[19]. M. Fowler. (2011) PolyglotPersistence. [Online]. Available:” http://martinfowler.com/bliki/Polyglot Persistence.html”

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

[21]. S. Vinoski, “Serendipitous reuse,” IEEE Internet Computing, vol. 12, no. 1, pp. 84–87, Jan. 2008.

