Rethinking 'Things' - Fog Layer Interplay in IoT: A Mobile Code Approach

Behailu Negash, Tomi Westerlund, Pasi Liljeberg, Hannu Tenhunen

Rethinking 'Things' - Fog Layer Interplay in IoT: a Mobile Code Approach

Behailu Negash[1], Tomi Westerlund[1], Pasi Liljeberg[1], and Hannu Tenhunen[1,2]
Email: {behneg, tovewe, pakrli}@utu.fi, hannu@kth.se

[1]Department of Information Technology, University of Turku, Turku, Finland
[2]Department of Industrial and Medical Electronics, KTH Royal Institute of Technology, Stockholm, Sweden

Abstract. A client-server architecture style is one of the common approaches enabling separation of concerns in distributed systems. In the Internet of Things architecture, this approach exists in different configuration of sensors, actuators, gateways in the Fog layer and servers in the Cloud. This configuration affects the degree of interoperability, scalability and other functional and non-functional system requirements. In this paper, we reflect on best practices in the web and REST style to address IoT challenges; one of the constraints in REST, Code on Demand, is used for IoT to enhance the flexibility and interoperability of resource constrained clients at the perception layer. Scripts written in a domain specific language,DoS-IL, are organized and stored at the Fog layer for sensor and actuators nodes to request and execute the incoming script. A generic application layer protocol and RESTful server are presented along with experimental results.

Keywords: Internet of Things, Architecture, Interoperability, Fog computing, Scalability, DoS-IL, Programmability

1 Introduction

The growing number of cheaper, smaller and embedded devices connected to the Internet is fueling the development of what is known as the Internet of Things (IoT). IoT is expected to bring billions of devices online enabling a wide range of possibilities for everyday life. This has two main perspectives merged as one: the connectivity of the devices and the application running on top of it. In contrast to the development of the current Internet and the World Wide Web, IoT is developing as one it is creating confusion in naming and its vision [1]. This paper reflects on previous approaches and techniques used to address the challenges faced during the development of the Internet and the Web to learn from and adapt it for IoT. The World Wide Web is planned to be a virtual space where people can interact and information can be stored [2]. This is enabled by the underlying interoperable connectivity provided by the Internet. However, before reaching such level of scale and interoperablity, both the Web and the Internet

has passed through competing heterogeneous platforms and protocols. The lack of interoperability is even worse in the Internet of Things; diverse communication protocols, architecture, data format and middleware exist. This limits the level of connectivity possible in IoT, and hence the application running over it are usually vertical silos. To bridge the silos and build a global system, a clear IoT vision is required to move forward.

The IoT vision is still evolving and there are contributions from academia, industry and government institutions [1]. Singh *et al.* [3] summarize the IoT vision in three parts; the Internet vision, Things vision and Semantic vision. In general, an IoT vision can be put as smart objects capable of sharing meaningful information over the Internet. This is inline to the vision of the Web, to be shared information space for machines and people [2]. The main contribution of this paper is towards the realization of this vision via the extended architecture and tools discussed here. We start with a general discussion of the nature of the web as the largest distributed system and learn from it. Distributed systems is a general name used to refer to systems that run in multiple separate boundaries, physical or logical, and communicate to accomplish their task. Part of these systems naturally demand for physical separation of its components on to multiple devices - the Web and IoT are typical ones; a server component provides service for a client connected through the Internet. The role of the server in formatting (HTML), and locating (URL) the information requested, and the standard of the communication (HTTP) are the driving reasons for the success of the Web.

An architectural enhancement to the original design of the web came through REST (Representational State Transfer) architecture style [4]. One of the optional constraints put in this style is the code on demand (COD) approach. COD is the process of sending a code for a client to execute, which is a specific case of mobile code pattern [5]. In this paper, we explore mobile code style for the Internet of Things through a semantically organized set of scripts and a novel server in the Fog layer to enable interoperability and enhance the scaling of IoT towards its vision. Devices embedded in physical objects send requests to the server in the Fog layer for instructions on how to present their sensor data or alter built-in actuators. An IoT-lite ontology [6] is used to annotate the devices in our experiment and DoS-IL (Domain Specific IoT Language) [7], an IoT domain specific language is used for the scripts. In summary, the main contributions of this paper are:

- Semantically organized domain specific scripts
- An abstract application layer protocol for communication
- REST like uniform interface and a service bundled together

The rest of the paper is organized in the following manner. Section 2 present the challenges in the Internet of Things, a motivation for this work and state of the art in this area, followed by details of the main work of the paper in Section 3. The results of the implementation and evaluation of the performance are reported in Section 4. The final section concludes the paper with the summary of results and discussion of planned continuation of the work.

2 Motivation and Challenges

This research aims to address some of the challenges facing the IoT, such as reconfiguration, interoperability and scalability. To clarify on these difficulties and motivate the work, we consider a simplified IoT use case in a remote patient monitoring in a smart home system and highlight the integration challenges and proposed potential solutions for such systems. There exist multiple challenges to reach an Internet scale and interplay with the state of the art method. Most IoT systems exist as vertical silos forming boundaries of device architecture, programming approach, network protocol and data formats among others. Each application works independently and is connected to the Internet allowing users to interact remotely. The majority of these devices are battery powered, has small memory and limited processing power. Moreover, the network interface associated usually has lower bandwidth and follow different protocols. Looking at the 'things' vision of IoT [3], one expects to have a seamless integration across these boundaries with proper authentication. Similarly, the 'Internet' vision promises to deliver an Internet scale system of smart systems. Some of the challenges, such as scale and heterogeneity, hindering the realization of this vision are presented by Zorzi *et al.* [8] along with a proposed solution from protocol perspective. Another work presented in [9] shows a horizontal architecture for IoT to help manage the challenge of interoperability and ease of programmability using a software defined networking scheme. At the highest level of abstraction of the systems, the data format and semantic knowledge of the exchanged information has to match for the systems to interoperate. Moreover, the architecture of the systems is a key component for integration. There has been many contributions from industry, academia and public projects to close the gap and hide the heterogeneity. An open survey shows some of these challenges and solutions [10]. One of the proposed solutions is IoT-A [11] - a project aimed to address these challenges with a reference architecture. Another approach is using middleware to bridge the gap in such systems. Razzaque *et al.* [12] present a survey of some of the middleware proposals. In the following sections, we highlight our approach in addressing this challenges and present the experimental results obtained.

3 Enabling code on demand

The introduction of Representational State Transfer (REST) as an architectural style in the web simplified the development and consumption of services in distributed systems across the globe. One of the constraints in REST in the area of mobile code architecture style is Code on demand (COD) [4]. It enables a client node to extend its feature through an executable code sent from the server. Code on Demand is one of the ways code mobility is achieved. Fuggetta *et al.* [5] discuss some of the benefits and uses of mobile code. Our approach resembles the case of remote device control and configuration in [5], where we allow the reconfiguration of devices in the perception layer with DoS-IL [7]. A script stored in the gateways at the Fog layer will be sent to a device on demand to be executed.

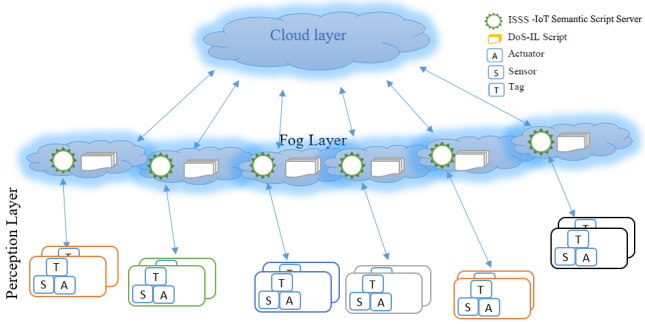

Fig. 1. High level deployment architecture of the proposed system

The result of the execution is also transferred to the gateway via a generic application layer protocol running over heterogeneous network interfaces. To allow this, the overall architecture of the system and its details are discussed in the following subsections.

3.1 System architecture

Architecture plays a critical role in achieving the desired functional and non-functional requirements of a system. Currently there are two main approaches of connecting devices to a gateway, regardless of the communication protocol: writing the program to read sensors, format, process and send it, or make a service to listen and handle incoming requests. This is similar to push and pull form of communication. In both cases, the node is rich in feature, it has the resources needed as well as the know how to manage it. The role of the gateway in the Fog layer is to provide connectivity and handle incoming or pulled data from the perception layer. In these approaches, maintenance of the application becomes a challenge after deployment. Moreover, the approaches are not scalable to the Internet level as in the case of the Web. For the second case, the nodes are usually resource constrained (processing power, battery or memory) to run a service that is available at anytime. Our approach is different in that the nodes at the perception layer still contain the resources (sensors, actuators or tags), but lack the know how which is the main point of functional or non-functional changes. This know how is written and stored in the Fog layer and sent to the node on a GET request. Figure 1 shows a generic three tier deployment architecture of an IoT system and the different components mapped on it.

3.2 Script organization

The scripts that are stored in the server are written using a domain specific language called DoS-IL. Each one of these scripts has a name and address that is resolved from the domain knowledge represented via an extended annotation

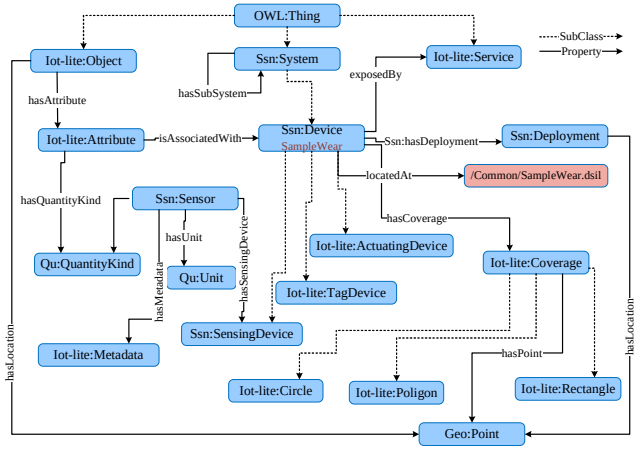

Fig. 2. IoT-lite ontology modified for the server

using IoT-lite ontology[6] as shown in Figure 2 . Devices have unique names that gets resolved into a URL of the location of the script using a sparql query from the ontology. The ontology is formatted in RDF (Resource Description Format) and stored in the gateway representing part of the Fog layer. This enables easy replication and merging of the ontology in other gateways when necessary. To parse and query the ontology, rdflib - a python library is used for sparql parsing. In a hierarchical Fog architecture, as proposed by the OpenFog consortium [13], the segments of the ontology can easily be replicated to multiple nodes for easy scaling of the system. This will be part of our future extension of this work towards demonstrating our proposal over distributed hierarchical gateways.

3.3 Generic Application layer

One of the components that contributed to the success of the World Wide Web, besides the addressing means and the markup language, is Hypertext Transfer Protocol (HTTP). As an application layer protocol, it runs over the standard TCP/IP protocol. However, in Internet of Things, the majority of the devices, especially those in the perception layer, use various low power communication protocols. Our semantic server is designed to handle incoming requests encoded with this generic application protocol over heterogeneous network standards. An abstract class representing network interfaces is defined, which needs to be implemented for each supported network protocol. Request and response headers are only few bytes to work with networks of low bandwidth. The request header is two bytes long and it is organized as follow: first four bits indicate the version number, followed by two bits for the verb of the request (REST verbs), one bit to show what type of script format the client accepts and one status bit for notification of script changes. The second byte is for payload size, which is set

to 0 for GET requests. In case of the response, the response code takes the first byte followed by the remaining packets in the message and the checksum - a total of three bytes. The response codes and the header formats are shown in detail in the source code repository [14].

3.4 IoT Semantic Script Server

A uniform interface is defined for the communication channel between the clients and the IoT Semantic Script Server (ISSS). Like a RESTful service, a GET request is used for the request of the configuration script while POST is used to send the data from the device for processing at the gateway. The method (GET or POST) is specified in the request header of the generic application layer protocol. Depending on the header, a specific request handler is passed the request to process it. Whenever a GET request is received, the request handler resolves the requested name from the IoT-lite ontology to a proper script location and fragment the script based on the available bandwidth. This server instantiates and listens to all the available network interfaces through the concrete implementations of the abstract class. Regardless of the underlying network protocol, the function of the server and the application layer protocol remains the same. The details of the implementation of our server and its components are shown in more details in the following section.

4 Demonstration and Evaluation

To validate our proposal beyond a conceptual point, we have developed a simplified scenario using our python based server implementation. Figure 3 shows the setup of our demonstrator system. As shown in Figure 3, the left side, Arduino Mega, is the client node representing the perception layer. The server, Intel Galileo Gen2, is shown in the right hand side running a Linux distribution running on SD card provided with the board. The source code of the server implementation is also available online [14]. To evaluate the performance of our service, the following parameters are used to measure the communication and computing overhead for 100 iterations. First, the round trip time of a simple message is recorded, shown in blue as round trip reference in Fig. 4. Then, we measured both the time taken to send as well as the total time taken to request and receive a script of size 527 bytes (shown in gray and red in Fig. 4). In summary, the total additional time of sending the script compared to the reference value (average of 73 ms) grow approximately six times to an average of 429 ms subject to the experimental setup. A second script is also tested and shown in yellow. Furthermore, Listing 1.1 shows how the script is fragmented and the response header; 100 for continue followed by the remaining packets, the checksum and part of the script. The time required to resolve name to url at the gateway is also analyzed. It takes longer on the first time of parsing the rdf (almost 238 ms) and subsequent calls take only few milliseconds (42 ms).

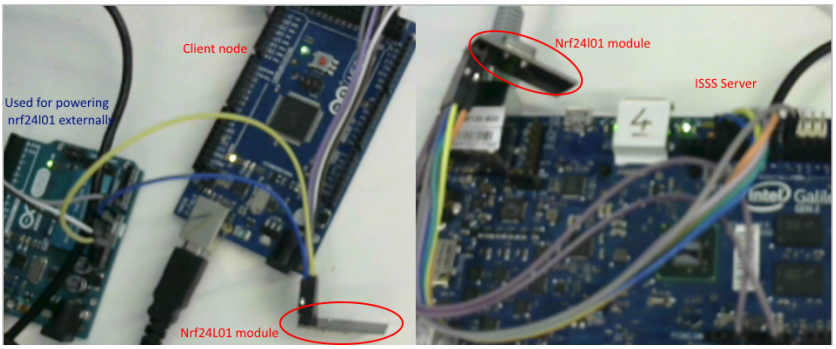

Fig. 3. Simplified demonstrator implementation

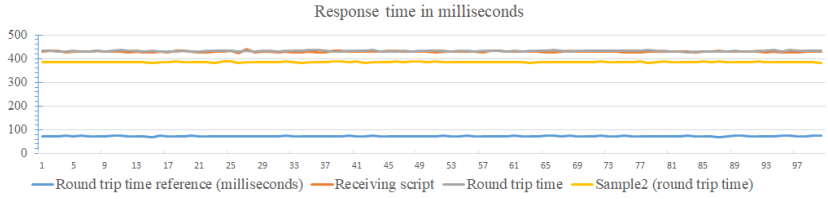

Fig. 4. Response time analysis of the semantic server

Listing 1.1. Partial Script fragmentation

```
 dss/common/SampleWear.dsil
[100, 20, '52', '!Version 0.1!\r\nwhile ( dev']
[100, 19, '9d', 'ice ->ready());\r\n\tdevice -']
[100, 18, 'e8', '>every(12){\r\n\t\tsnumber _si ']
[100, 17, 'f3', 'd:=4;\r\n\t\tsnumber _aid:=2;\r']

[100, 2, '32', '\r\n\t\t\telse\r\n\t\t\t{\r\n\t\t\t\tdevic']
[100, 1, '82', 'e ->sleep();\r\n\t\t\t}\r\n\t\t}\r\n}']
[200, 0, 'c9', '\r\n\t\r\n']
```

5 Conclusion and Future works

With the ever increasing number of network protocols, platforms and data formats used in the Internet of Things, the vision of realizing an integrated IoT system is getting difficult. This work refers back to the beginning of the World Wide Web to learn ways of overcoming heterogeneity and reach an Internet level of scaling. We implemented an IoT Semantic Script Server listening over a generic application protocol and organized it using a lightweight IoT ontology. An implementation of the generic network interface for nRF24L01 based network is developed and performance evaluation is carried out. The evaluation shows the fragmentation of the script for sending over a low bandwidth network, the round trip time and the performance of the process of resolving a device name to a script path. The initial version of the application layer protocol is

also very light and optimized for code on demand approach. In summary, this paper introduced a novel architecture for IoT, with the help of a domain specific language, that enables both semantic and syntactic interoperability and facilitate the growth of IoT to an Internet scale. To further extend this work, we plan to make the service implementation to listen to multiple network interfaces and work with distributed script locations over multiple Fog gateways. Furthermore, we believe that inclusion of standards from some of the big consortium make our efforts comprehensive in pushing forward the integration effort of IoT. One of this standards considered for future works is the data format standard from the open interconnect consortium.

References

1. R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the internet of things (IoT)," IEEE, Technical Report, 2015.
2. T. Berners-Lee, "Www: past, present, and future," *Computer*, vol. 29, no. 10, pp. 69–77, Oct 1996.
3. D. Singh, G. Tripathi, and A. J. Jara, "A survey of internet-of-things: Future vision, architecture, challenges and services," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, March 2014, pp. 287–292.
4. R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.
5. A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, May 1998.
6. M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor, "Iot-lite: A lightweight semantic model for the internet of things," in *2016 Intl. IEEE Conferences on UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld*, July 2016, pp. 90–97.
7. B. Negash, T. Westerlund, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "Dos-il: A domain specific internet of things language for resource constrained devices," *Procedia Computer Science*, vol. 109, pp. 416 – 423, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050917310876
8. M. Zorzi, A. Gluhak, S. Lange, and A. Bassi, "From today's intranet of things to a future internet of things: a wireless- and mobility-related view," *IEEE Wireless Communications*, vol. 17, no. 6, pp. 44–51, December 2010.
9. Y. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani, "A sdn-based architecture for horizontal internet of things services," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–7.
10. Eclipse IoT Working Group, "IoT developer suvey," IEEE, IoT Eclipse, Agile, Technical Report, 2016.
11. IoT-A Project, "Internet of things - architecture, IoT-A, deliverable d1.5 - final architecture reference model for the IoT v3.0," EU-FP7, Technical Report, 2013. [Online]. Available: http://www.iot-a.eu/public/public-documents/d1.5/view
12. M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, Feb 2016.
13. OpenFog Consortium, "OpenFog Reference Architecture for Fog Computing," OpenFog Consortium, Technical Report, 2017.
14. B. Negash. ISSS implementation. https://github.com/behailus/ISSS.