Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2016) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017)

# DoS-IL: A Domain Specific Internet of Things Language for Resource Constrained Devices

Behailu Negash[a,*], Tomi Westerlund[a], Amir M. Rahmani[b,d], Pasi Liljeberg[a], Hannu Tenhunen[a,c]

[a] *University of Turku, Department of Future Technologies, Turku, Finland*
[b] *University of California, Department of Computer Science, Irvine, California, USA*
[c] *KTH Royal Institute of Technology, Department of Industrial and Medical Electronics, Stockholm, Sweden*
[d] *TU Wien, Institute of Computer Technology, Vienna, Austria*

**Abstract**

The common approach enabling a resource constrained device to get connected to the Internet is through programming instructions and transferring it to an embedded device. This procedure involves various tools and cross-compiling of the code depending on the platform architecture. In practical IoT applications, where a huge number of nodes exist, this process becomes almost impossible due to the heterogeneous platforms and protocols involved and the deployment conditions. This paper introduces a flexible and scalable approach that enhances modifiability and programmability through client-server-server-client architecture. It allows changing the behavior of the system after deployment through a lightweight script written with a domain specific language, DoS-IL, and stored in a gateway at the fog layer. An embedded resource browser is used to request and execute the script. The results of analysis for this model and the tools developed along the way are discussed.

Peer-review under responsibility of the Conference Program Chairs.

*Keywords:* Internet of Things (IoT); Architecture; IoT-A; Programmability; Interoperability; Domain Specific Language; Scalability

## 1. Introduction

The coke machine of the students of Carnegie Mellon University (CMU) [1] has evolved, reformed and introduced as the Internet of Things (IoT) to dominate the technology industry after three decades. The students from CMU enabled a coke vending machine to connect to the network and publish availability and temperature of soda. Since the past five years, it is common to hear similar stories with multitude of gadgets capable of connecting to the Internet publishing their status. These consumer products come with different architecture and communication protocol and programmed to work usually in pre-configured ways. Updates are pushed from the manufacturers server to these devices when available and in most cases the devices send the data to a cloud service built by a service provider. Regardless of

* Corresponding author. Tel.: +358-456-014-324 .
*E-mail address:* behneg@utu.fi


Peer-review under responsibility of the Conference Program Chairs.

the purpose of the system, a generalized definition of the Internet of Things is an autonomous network of devices embedded in a physical object that sense internal or environmental changes or modify the state and communicate through the Internet at anytime[2]. It is estimated that there will be tens of billions of such devices connected to the Internet in the coming couple of years. This inter communication and access to the Internet enable them to be intelligent and fuel a wide range of applications. IoT is expected to transform the current processes in many aspects of life[3]. With so many potential prospects coming with IoT, there are many challenges that need to be addressed.

The trend in computing is moving to cheaper, smaller and faster devices in general. This is one of the enablers of the progress of IoT and the ubiquitous nature of the embedded devices in physical objects. However, it comes with its own challenges; most of these devices are resource constrained to function in the traditional way of localized processing and storage of information. These devices are constrained in the amount of memory available for storage, their processing power, the communication bandwidth and the available electrical power (battery or harvested). Moreover, the dominant mode of communication in IoT is wireless and there are already diverse variety of such protocols. In addition to protocol differences, there are platform and format variations that limit the integration possibility. This differences force system developers to utilize a range of tools and require various skill set to program. For instance, interoperability middleware[4] are common solutions. Looking back at historical approaches of distributed computing, in response to resource constraints, client devices have gone through changing work load in computing and programming methodologies. Gradually, the approach has shifted in the world wide web to a more protocol agnostic client server architecture where by generic client side browsers request for information and parse the incoming data to present it to the user.

With the introduction of billions of devices in Internet of Things, the client server configuration of embedded devices and the Cloud has been modified with an intermediate Fog computing layer[5]. This layer is in an ideal position to support the limitation of the embedded devices in the lower layer, or usually referred to as the perception layer. It provides low latency communication with real time feedback and bring computing, storage and networking services close to the embedded devices. This layer is characterized by its wide geographical distribution, location awareness, and physical proximity that can be leveraged to provide a wide range of services. This paper utilizes the Fog layer to enhance programmability in the perception layer by introducing an IoT domain specific language that can be used to write a script to be stored in the Fog layer and accessed by devices in the perception layer. The main contributions of this paper are:

- A domain Specific IoT Language (DoS-IL) for resource constrained devices
- A Device Object Model (DOM) that is manipulated by DoS-IL
- A lightweight interpreter of DoS-IL that sits in an object browser

Prior to going into the details of the implementation of the language, interpreter and resource browser, we highlight the limitation in IoT to motivate the importance of DoS-IL. In Section 2, the challenges of IoT are revised and show the need for novel approaches of programmability. In subsequent sections, the general architecture of the system is presented (Section 3), DoS-IL is described in more details and detailed analysis is presented (Section 4). The analysis of the work is presented in Section 5. The last two parts present other related IoT domain specific languages and researches that enhance programmability, scalability and interoperability of IoT, in Section 6, and the concluding remarks in the final section.

## 2. Challenges in IoT

To identify the challenges in IoT and motivate the need for novel approaches, consider a scenario where an IoT application is composed of multiple devices with different platform architecture, using various communication protocols, consisting thousands of devices. A developer starts by selecting the best fit embedded operating system or can choose to work without one. In the condition, there are also wide range of embedded operating systems to choose from based on the support for the hardware platform architecture and the communication protocol. After picking the best fit from all these, the actual programming of the logic is dependent on the selected operating system, protocol and platform. The program gets cross-compiled in a host computer and transferred to the end device. Given the nature of IoT systems, the devices get deployed in remote locations or embedded in physical objects. In more advanced cases,

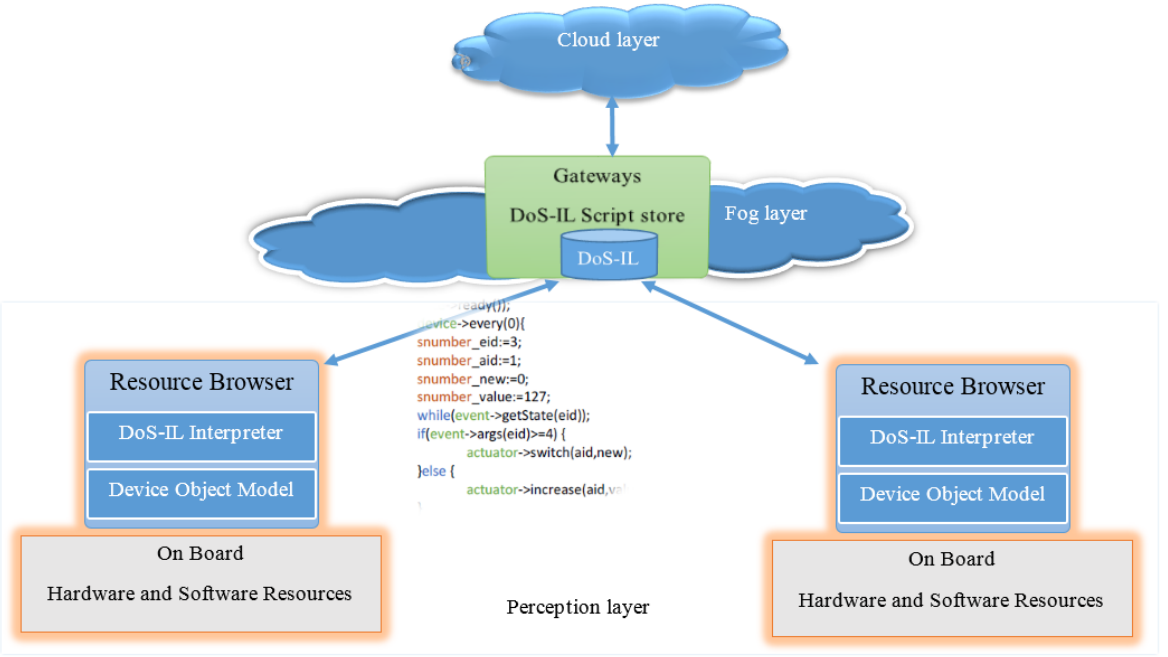



Fig. 1: Architecture overview of IoT with DoS-IL

it can be deployed in locations where physical access is hardly possible. In such scenarios, to bring back these devices to modify part of the programmed logic is impractical. In addition, looking at the scale of deployment expected, going through the above process for each device is tedious and almost impossible. The diversity of the communication protocols and data formats exacerbate the problem by adding the dimension of heterogeneity. There are many efforts in the area of interoperability by introducing a middleware[4] to hide the underlying platform and protocol variations. Some of these middleware solutions are challenged by the lack of enough resources to host the additional overhead in memory and computing needs.

The solution proposed in this paper is inspired by earlier approaches in distributed and networked systems. During the early days of networked and distributed systems, similar challenges at different scales have been well researched. To explore similarities and extract best approaches, we reflect on earlier approaches. Tanenbaum and Van Renesse[6] discuss distributed system and networked system and highlight the design issues in each category. Moreover, Fielding[7] provides an overview of the design challenges of the web architecture; some of these challenges presented are extensibility and scalability. Sir Tim Berners-Lee[8] explains the time before the introduction of the world wide web as full of heterogeneous networks, disk formats, data formats and character encoding. The design goals of the web has been to overcome these challenges and simplify information sharing. Fielding[7] in his doctoral dissertation, where he discuss the REST (Representational State Transfer) architecture style, also presents the approach followed to overcome the challenges. The current IoT programming and deployment is at a comparable stage, in terms of heterogeneity and programming approach, to the days before the web. This paper introduces an approach to include application control script at the Fog layer to scale and extend IoT systems. The next section gives an elaborated architectural aspect of DoS-IL.

## 3. System Architecture

One of the approaches to connecting a sensor or actuator device to the Internet of Things has been through a direct cloud communication. However, this approach has brought many challenges and is limited by the available of local resources. The communication with the cloud has higher latency, requires more energy and supports devices

capable of IP based communication. However, as mentioned previously, the majority of IoT devices are dominated by low power wireless protocols, some of these protocols are not IP based, their memory is constrained and the system requirement demands for real-time or low latency feedback. These IoT needs with the huge number of sensors and actuators joining the network, Fog computing has been identified as an enabler. In this paper, the role of the Fog computing layer is not simply as a sink layer for the data coming from the sensors in the perception layer but it is a fundamental server layer that avail the program logic as shown in Figure 1. This forms a client-server communication between the perception and Fog layer. Looking up to the Cloud layer from the Fog layer, regardless of the capacity of the layers, the two physical layers complement each other. Users of the system mostly access IoT applications through the Cloud, to have a comprehensive knowledge of the system across multiple geographically distributed Fog layer devices. The system can also be accessed through local gateways in the Fog layer, for localized information and notifications from the system. This architecture forms a fusion of two client-server arrangements effectively compose to what is referred in this paper as client-server-server-client form.

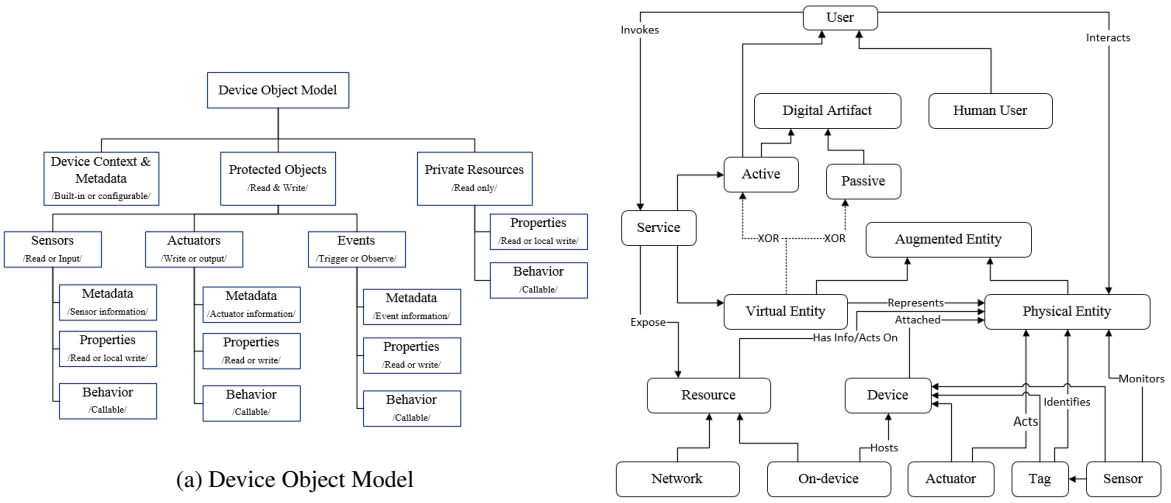
The overall architecture proposed in this paper is depicted by Figure 1. A three layer physical and functional separation of IoT devices is shown. The main role of DoS-IL, as shown in the figure, is between the Fog computing layer and the resource constrained devices in the perception or sensor layer. Scripts written in DoS-IL are stored in a gateway in the Fog layer, for easy maintenance and modification after deployment, without the need to physically access the end nodes. The scripts will be organized based on the target device unique id, or group of functionally similar devices. Version information of scripts is modified to reflect corresponding updates for the target devices to pull the latest one. The gateway serves the script to the end node only when there is a new version or a new device requests it. On the target node, the Device Object Model, discussed in Section 4.1 exposes the available resources for the DoS-IL script to manipulate. The script is interpreted by a DoS-IL interpreter, embedded in a resource browser of devices in the perception layer.

## 4. IoT Domain Specific Programming

A domain specific language (DSL) is a specialized form of programming language for a specific application domain to simplify the process of implementation. In contrast, general purpose programming languages are generic and can be used to solve a range of problems in many application domains. Even though a domain specific language is limited in its usage outside the domain, it can be used to write efficient code to solve problems in the target domain. There has been a range of DSLs for IoT, to enhance the programmability or simplify the process of wiring different components of an IoT system. Some of these DSLs are discussed in Section 6. Before discussing the difference, the following subsections provide an overview of DoS-IL and its components.

### *4.1. Device Object Model*

The concept of the Device Object Model (DOM) in this work is influenced by the Document Object Model[9]. Document Object Model is a platform and language agnostic interface exposed so that programs can manipulate its elements. It helps in formatting the presentation of the content, while enabling dynamic way of updating the content. The Document Object Model of an HTML document is built using HTML elements hierarchically and JavaScript can be used to manipulate it at run-time. For example, the root element of an HTML document has head and body, where the body can have elements like Div or H1. When a client requests for a content from a server, the information comes structured in Html along with a JavaScript code to dynamically change the format. Analogous to this flow of events, the resource constrained clients can be represented using Device Object Model (DOM). The DOM in this work represents the resources that are available in the constrained device; these resources can be hardware resources or those shared with the external world through the network such as sensor reading or private resources of storage, memory or battery power. The DOM organizes and exposes the resources available in the device. Figure 2a show the initial device object model created to work with DoS-IL. There are three main children under the DOM: First, private read only resources that constrain the device, such as network protocol, memory and battery values are in this category. The middle category contains the functional components of the device, such as sensors, actuators and events. The last category identify the device and provide other relevant information such as context, location and environmental conditions, and identifier tags, if available.




(a) Device Object Model

(b) IoT domain model

Fig. 2: Domain model of IoT and its DOM representation

The protected elements of the Device Object Model are the main elements that are manipulated to change the behaviour and function of the device. It contains, Sensors, Actuators and Events that occur in the device. Consider a certain device node interacting in an IoT system; there are external devices interested in the sensor values and it provides this information, or it is interested in reading values from another device so that to act using the local actuator or it listens to other external or internal events. Each of the three main elements have properties, behaviours and metadata information represented in the DOM. The metadata about these elements contain their unique identifiers or time-stamp of the last occurrence. The DOM of a device is exposed and is manipulated with a script written using DoS-IL. In the following section, the scripting DSL is explained in more details.

### *4.2. Domain Specific Programming Language*

There are a set of basic operations programmed in IoT devices. According to the Information and communication model presented in the IoT architecture model, IoT-A [10], there are certain abstractions that help design our DSL. Looking at the simplified IoT domain model [10] in Figure 2b, a physical device is supported with sensors, actuators and a tag to form an augmented entity. These elements have distinct functions and belong to the same class, shown as Device. In a similar fashion, IoT-A provides the communication model followed by these entities; user - service interaction, service - service interaction, and service - resource - device interactions. These can be summarized as Request, Response, Publish or Subscribe. Publish can be used for reading sensor information and sending to a remote destination, Request for information from remote location which is answered by a Response, and Subscribe for notification of an external event. As a response to these communication requirements, the device acts in different ways. For example, to handle Publish type of communication, the device is programmed to read the sensor data and send the information at specified time intervals. This program is written in a general programming language and cross-compiled, and uploaded to the device. Once the device is deployed in a certain location, modifying the code will involve going through the process again or using other device management protocols [11] to handle the update process. Similarly, an actuator that acts according to the information provided by a sensor receives the information either from a remote sensor or a local one and operate its function. To demonstrate the difference of our IoT DSL, consider the following sample code (shown in Figure 3) written with DoS-IL. The function of the code is to wait until the device initializes the DOM and loop continuously every 12ms to check the reading of sensor identified by unique id, for instance 4. If its value exceeds a certain value (12.35) manipulate the actuator (rotate) or otherwise wait for an event to fire. When the event fires and the value of the argument is negative, it sends the sensor value to the server and finally the device goes to sleep mode. This code is written and stored in the DoS-IL store in the Fog layer, as shown

```
1
2  !Version .1!
3  while ( device ->ready());
4      device ->every(12){
5          snumber _sid:=4;  snumber _aid:=2; snumber _eid:=16;
6          sformat _format := {id, timestamp, unit, value, type };
7          sval _measure := sensor ->read(sid);
8          if (measure->Value >= 12.35){
9              actuator ->rotate(aid,90);
10         } else {
11             while ( event ->getState(eid));
12             if( event ->args(eid) < 0)  {
13                 device ->accumulate(measure,format);
14             } else {
15                 device ->sleep();
16             }
17         }
18     }
```

```
1
2  !Version 0.1!
3  while ( device ->ready());
4      device ->every(0){
5      snumber _eid:=3;
6      snumber _aid:=1;
7      snumber _new:=0;
8      snumber _value:=127;
9      while ( event ->getState(eid));
10     if ( event ->args(eid)>=4) {
11         actuator ->switch(aid,new);
12     } else {
13         actuator ->increase(aid,value);
14     }
15 }
```

Fig. 3: Sample DoS-IL codes

in Figure 1. The resource browser in a sensor node, or a board with sensors and actuators, access this script and the interpreter executes this code. The following section describes the resource browser to access such a script from the Fog layer.

### *4.3. Resource Browser*

The resource browser is integrated as part of the main software platform of the device. It integrates an interpreter of DoS-IL and manipulates the DOM based on the instruction coming from the Fog layer. It is a generic application software layer that discovers[4] a near by Fog layer device and requests for the script to execute. This browser and its integrated interpreter are lightweight enough to fit in resource constrained devices. The interpreter is self contained in this initial version; it has a tokenizer to break the instruction into smaller tokens, label them according to DoS-IL grammar, group the tokens to instructions and expressions to execute them. However, to make it more lighter, the pre-processing task can be done by the Fog layer and the function of the browser will be to simply execute the expressions and instructions. The script maintains a hierarchy that makes it easy to execute in the browser. A DoS-IL script has two main parts; set-up or check and repeat sections. Sections contain blocks that organize statements.

## 5. Analysing DoS-IL

It was briefly mentioned earlier that there exist two main sections in a typical DoS-IL script. The setup section makes sure the device is initialized and the DOM is composed from the available devices locally (up to line 4 in Figure 3). The remaining section executes after a successful initialization of the device. Consider the code listing in Figure 3 with the DOM structure presented in Figure 2a. At the start of the analysis phase, the interpreter identifies the DOM elements from the language construct. In this sample script, it assumes that the device considered contains several sensors and actuators built into the same board. However, the case of individual sensor or actuator per board is trivial (second code in Figure 3). There are four DOM elements used in this sample script; these are device, sensor, event and actuator. The device element is the parent of the rest and contains behaviours that alter the whole platform. Referring the DOM in Figure 2a, both device context and private resources are exposed via the device element. DoS-IL defines few types to handle various categories of variables; snumber, sformat, and sval representing numerical data, data formatting and values with formatting respectively. In addition, there are grammatical words that determine the flow of events, conditionals and wait statements.

The interpreter splits the script into tokens first, identify the function of each token and structure it in a convenient way for execution. The above script in the left, for instance, is broken into 125 tokens first and each one is labeled with one of the following classes: DOM element, type, variable, constant, language construct, or operator. For instance, looking the code list above, 'while' is classified as language construct, '(' as operator, 'device' as DOM element and 'ready()' is related to device as part of the DOM element. These tokens are organized into building blocks of the executable script into blocks, statements and expressions. A script has two sections, a section has blocks, a

block has statements or block of statement and a statement has expressions. For instance, the left script of Figure 3 forms 15 statements, 28 expressions and 5 blocks. There is a limitation to the number of expressions allowed in a statement in this version. Unlike sections, statements and expressions that vary based on their function, all blocks are functionally similar. Analogous to the functional differences in sections, there are four types of statements; wait, DOM call, assignment and conditional statements. The expressions are also labeled as variable, constant, call or declare expression. From the example code above, the first line of the script is a wait statement containing a call expression inside it to check if the device is ready. Once the statements, blocks and expressions are identified, the execution is straight forward.

The script is executed sequentially from the beginning, except in conditional statements where it has to jump to the appropriate location. The interpreter has a pointer to the tokens, which serves as program counter and conditional statements move the pointer to the right location. It also has a stack to store variables and values declared in the script. The size of the DSL interpreter embedded in the resource browser is only **76KB** in size. To compare the performance of this architecture and our DSL, consider an alternative way of managing application modification after deployment. For instance, the Open Mobile Alliance (OMA)[11] has a lightweight M2M solution that benefits in device management. OMA lightweight has client and server side solutions that collaborate to deliver updates among other advantages. However, it relies on CoAP and excludes many devices in IoT that communicate via non-IP based networks. In contrast, DoS-IL can be transferred in any communication protocol and the size of the script is very small due to the compact design of DoS-IL. For example, the script shown in Figure 3 is only **527 bytes** and **292 bytes** for the left and right side respectively. This can be easily transferred through the limited bandwidth available in many low power network protocols in one packet. Moreover, achieving a web like structure opens a wide range of possibilities to scale. Table 1 summarise the language constructs of DoS-IL. The tokens are organized in three categories, key words that determine the execution flow, operators (partial) that help in simplifying writing code and DOM elements.

Table 1: DoS-IL language construct and DOM interface

| Category | Tokens | Description |
|---|---|---|
| Key words | while, if, else | Flow of execution, jump or wait |
| Operators | =,–>,< , > , := , _ , {, ( & (Partial list[12]) | Comparison, assignment operators, logical and separators |
| DOM elements | device, sensor, actuator, event, resource | DOM interface, behaviors, properties and metadata |

## 6. Related works

Domain Specific Languages (DSL) are common in enterprise application areas. DSLs deal with problems of a specific domain. For instance, SQL (Structured Query Language) - a DSL to manage data in a relational data store and HTML (Hyper Text Markup Language) - a language to create and represent the structure of a web document, are very well known. In the domain of IoT, there are few proposals of DSL to overcome the challenges in IoT. Most of these DSLs are proposed to help with simplification of the process of creating the work flow. DSL-4-IoT[13] is an integrated development environment and graphical programing approach that targets to simplify the programming complexity of IoT and overcome heterogeneity. It generates the code that will be plugged into openHAB[14], which is a technology agnostic platform for smart homes. Even though it addresses interoperability and programmability challenges, DSL-4-IoT is limited in terms of the scalability and extensibility. Similar approaches have been presented in Node-RED[15] to simplify the programming approach. Eterovic et. al.[16] approach the programming of IoT systems through visual programming based on UML to enhance its usability for people without engineering background. The challenge of scaling and modifying IoT systems is not the focus of this work neither. A web based DSL for IoT is proposed by Sneps-Sneppe et.al.[17] to address IoT programming challenges.

Another DSL for IoT programming and integration with other services is IFTT[18]. There are service applets, building blocks that integrate to form certain functions. It makes the process of assembling functionality from social networks and other platforms easier. Midgar[19] is another graphical DSL for IoT to simplify the programming process and enable easier integration by non-developers. In comparison to these DSL proposals, DoS-IL has certain similarities and unique contributions; programmability is a common agenda in these proposals and DoS-IL. However, DoS-IL

targets to address scalability and modifiability or extensibility at the same time. As presented in previous sections, DoS-IL is not generated and installed on the target device during the development process; it is written and stored in the Fog layer and run on devices in the perception layer. Moreover, this work also introduces a device object model that simplify modification of application behavior at run-time.

## 7. Conclusion

The Internet of Things promise a wide range of benefits to humanity in different areas of life. Currently identified areas of application are just the tip of the iceberg. However, in its current state, IoT has multiple challenges that opened active research areas. Scalability, interoperability, extensibility and programmability are some of these challenges that are targeted by this work. Unlike the current way of programming resource constrained devices using high level languages that limit scaling, modifiability and scalability after deployment, this work promotes a reversed approach where the behavior of client nodes can be dynamically changed from gateways at the Fog layer. This paradigm change requires certain tools that are also introduced here. A platform independent Device Object Model (DOM) is used to organize and expose the resources in a device and DoS-IL is a domain specific language developed to enable such a paradigm change that comes with a lightweight interpreter and a resource browser. The memory footprint of the interpreter is only 76KB and it is the main component of the resource browser. The size of the script that will be written varies depending on the application logic; however, the sample scripts provided here show that it is compact enough (less than a KB) and small in general for transferring. The DOM interface exposed is generic and the implementation varies on the available resources in the target device. The source code of the interpreter, the sample codes and the DOM interface with a mock of the implementation is available on Github [12]. Development of the script server in the gateway and enhanced device management will be developed in subsequent future enhancements of the interpreter and the resource browser.

## References


1. CMU. The ”only” coke machine on the internet. http://www.cs.cmu.edu/ coke/history*long.txt*.
2. Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the internet of things (IoT). Technical report, IEEE, 2015.
3. Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Comput. Netw.*, 54(15):2787–2805, October 2010.
4. Behailu Negash, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. LISA: lightweight internet of things service bus architecture. In *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015), London, UK, June 2-5, 2015*, pages 436–443, 2015.
5. Flavio Bonomi, Rodolfo A. Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012*, pages 13–16, 2012.
6. Andrew S. Tanenbaum and Robbert Van Renesse. Distributed operating systems. *ACM Comput. Surv.*, 17(4):419–470, December 1985.
7. Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.
8. T. Berners-Lee. Www: past, present, and future. *Computer*, 29(10):69–77, Oct 1996.
9. W3C. Document Object Model (DOM). https://www.w3.org/DOM/.
10. IoT-A Project. Internet of things - architecture, IoT-A, deliverable d1.5 - final architecture reference model for the IoT v3.0. Technical report, EU-FP7, 2013.
11. Guenter Klas, Friedhelm Rodermund, Zach Shelby, Sandeep Akhouri, and Jan Hller. Lightweight M2M: Enabling Device Management and Applications for the Internet of Things. Technical report, Vodafone, ARM, Ericsson, 02 2014.
12. Behailu Negash. DoS-IL implementation. https://github.com/behailus/DoS-IL.git.
13. Adnan Salihbegovic, Teo Eterovic, Enio Kaljic, and Samir Ribic. Design of a domain specific language and IDE for internet of things applications. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2015, Opatija, Croatia, May 25-29, 2015*, pages 996–1001, 2015.
14. OpenHAB. Open source automation for smart home. http://www.openhab.org/features/introduction.html.
15. Node-RED. A visual tool for wiring the internet of things. http://nodered.org/docs/.
16. Teo Eterovic, Enio Kaljic, Dzenana Donko, Adnan Salihbegovic, and Samir Ribic. An internet of things visual domain specific modeling language based on UML. In *XXV International Conference on Information, Communication and Automation Technologies, ICAT 2015, Sarajevo, Bosnia and Herzegovina, October 29-31, 2015*, pages 1–5, 2015.
17. Manfred Sneps-Sneppe and Dmitry Namiot. On web-based domain-specific language for internet of things. *CoRR*, abs/1505.06713, 2015.
18. IFTTT. Qick integration with services. https://ifttt.com/about.
19. Cristian González García, Jordán Pascual Espada, Edward Rolando Núñez-Valdéz, and Vicente García-Díaz. Midgar: Domain-specific language to generate smart objects for an internet of things platform. In *Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2014, Birmingham, United Kingdom, 2-4 July, 2014*, pages 352–357, 2014.