

Article

Design and Implementation of Cloud-Centric Configuration Repository for DIY IoT Applications

Shabir Ahmad , Lei Hang and Do Hyeun Kim *

Department of Computer Engineering, Jeju National University, Jeju 63243, Korea; shabir@jejunu.ac.kr (S.A.); hanglei112233@hotmail.com (L.H.)

* Correspondence: kimdh@jejunu.ac.kr

Received: 30 December 2017; Accepted: 30 January 2018; Published: 6 February 2018

Abstract: The Do-It-Yourself (DIY) vision for the design of a smart and customizable IoT application demands the involvement of the general public in its development process. The general public lacks the technical knowledge for programming state-of-the-art prototyping and development kits. The latest IoT kits, for example, Raspberry Pi, are revolutionizing the DIY paradigm for IoT, and more than ever, a DIY intuitive programming interface is required to enable the masses to interact with and customize the behavior of remote IoT devices on the Internet. However, in most cases, these DIY toolkits store the resultant configuration data in local storage and, thus, cannot be accessed remotely. This paper presents the novel implementation of such a system, which not only enables the general public to customize the behavior of remote IoT devices through a visual interface, but also makes the configuration available everywhere and anytime by leveraging the power of cloud-based platforms. The interface enables the visualization of the resources exposed by remote embedded resources in the form of graphical virtual objects (VOs). These VOs are used to create the service design through simple operations like drag-and-drop and the setting of properties. The configuration created as a result is maintained as an XML document, which is ingested by the cloud platform, thus making it available to be used anywhere. We use the HTTP approach for the communication between the cloud and IoT toolbox and the cloud and real devices, but for communication between the toolbox and actual resources, CoAP is used. Finally, a smart home case study has been implemented and presented in order to assess the effectiveness of the proposed work.

Keywords: Internet of Things; wireless sensor networks; smart space; cloud computing; configuration management

1. Introduction

Humans by nature have a strong tendency to learn things by doing and trying to do things by one's self [1]. There are various drivers that push humans towards the Do-It-Yourself (DIY) approach. The major drivers for the DIY paradigm are creativity, simplification, extension, economic reasons and the need to control things [2]. Apart from these drivers, the recent advancements and innovations in DIY electronics are providing an opportunity for the masses to demonstrate their creativity. Systems on Chip (SoC), electronics development platforms and kits in the form of Arduino and Raspberry Pi are a huge inspiration for DIY. The simplicity and ease of development on these platforms are attracting more and more people towards DIY and, hence, enabling the general masses to express their creativity and genius.

According to a report from the International Telecommunication Union (ITU) in 2005, the "Internet of Things (IoT) will connect objects from the world, both in a sensory and in an intelligent way" [3,4]. IoT as it was perceived until the recent past had not been adopted by the masses [5], while the connections and devices in the IoT have grown in numbers since its advent [6,7], and with the

realization of the Industrial Internet of Things (IIoT) and 5G, the number is expected to increase exponentially [8]. This means that end-users' involvement in the IoT creation process is a crucial factor for its successful adaptation. According to the recently updated Gartner Hype Cycle [9], IoT is of great importance and presents the idea that IoT has come out of its imaginative and fictive stage [10] and now is considered as the real deal [11,12]. However, the end-users should be a part of the creation process while having the power to discover things [6,13], control it and effectively use the applications for smart environments [14]. The same idea is also presented by Xiao et al. [15].

In addition to the DIY driver, motivations and the need for massive end-users' involvement for the realization of IoT, the current makers revolution [16] is inculcating a new version of the DIY culture. Connecting things, people and ideas together is conceived as the term "making" [17]. The Internet is providing the bridge between the makers and the masses. Online communities of people not only share their ideas and creations, but also have a chance to communicate and help each other on a larger scale [18], thus eliminating the structural and technological obstacles in their way [19]. DiY Smart Experience (DiYSe) [20] presents a DIY manifesto of 13 statements focused towards the developers who design and implement digital creation systems for end-users. The manifesto also highlights the relation of DIY IoT to the maker movement.

Inspired by the maker movement, many researchers perceive IoT as not just the implementation of some proprietary services for a specific goal, but in fact, as a vast ecosystem of billions of devices [21], which present themselves as atomic services on the Internet. These services must be available for the masses to utilize them to make their custom solutions. This concept is termed as the Do-It-Yourself (DIY) paradigm, and its main motive is to reduce the dependence of common people on proprietary products. In other words, DIY is the technique or method of creating, modifying or repairing something without the direct aid of experts or professionals. As IoT envisions a global ecosystem of connected devices providing services to people, the DIY paradigm of development has been suggested as the prime solution towards the global realization and implementation of IoT. In the DIY scenario, for instance, the same general population that utilizes the services provided by the IoT implementation will be able to develop the IoT application according to their own needs and requirements. It is, however, a concern that the general population lacks the necessary technical experience and specifically the programming skills to develop IoT-related applications. This is why alternative development strategies are being investigated to provide people with DIY type development environments through which anyone, regardless of their programming skill level, can develop applications based on their own requirements [22]. In a nutshell, the relationship between IoT and DIY or maker communities are best conceived of as "Embracing IoT instead of Circumventing it", and this work is a step forward towards this vision.

Many researchers have put efforts in this regard and proposed various DIY-based architecture for easing up the development of IoT application and more specifically for enabling the general public to tinker with actual IoT resources to make a real IoT application; however, the DIY toolboxes in most cases use desktop technologies that would mean the configuration of the services composed would reside on the local system, and thus, there will be no way to access the configuration from any remote server, which in turn would mean that people shall not be able to control the devices outside their premises.

The contribution of this research is three-fold; firstly, we present a visual service designer with the drag-and-drop approach with the aim focusing on zero-programming customization of the functionality offered by remote and constrained IoT devices. The Constrained Application Protocol (CoAP) is among the most popular communication protocols for devices having limited resources. It is being standardized by the Internet Engineering Task Force (IETF) [23]. The proposed system utilizes CoAP resources as part of a CoAP server, which is implemented using the Raspberry PI board. Raspberry PI is the latest SD card size platform based on Linux distribution named Raspbian OS. The reason for selecting this platform is the growing acceptance of the platform for the development of IoT devices by the DIY community all over the world. These CoAP resources are shared by the

CoAP server with the visual service designer via a CoAP proxy. The proxy is a Java implementation based on the Californium [24] framework, which facilitates the discovery of CoAP resources and provides libraries for the generation of CoAP commands. The designer represents the resources as virtual objects. The user uses the graphical representations of the virtual objects to design a service flow. The service design is sent back to the proxy in the form of an XML document where it is parsed to extract the information regarding the customized behavior of the remote CoAP device. The proxy then generates CoAP commands using the Californium framework in order to operate the devices according to the new functionality defined.

We believe that it is not far in the future when such IoT devices will be commonly available through the Internet, and the non-programmer users will require such a DIY interface to customize the functionality of those devices. The focus of this work is not how to implement CoAP devices and program them as is the case for several implementations discussed in this section. The focus is rather to enable the general public to easily access remote devices and, based on the resources and functional capabilities exposed by these devices, provide an intuitive representation to the users in order to customize device's operations according to the users' needs. To our knowledge, the proposed DIY system is the first ever step towards the behavior customization of remote CoAP devices using a visual interface.

Secondly, in addition to the idea of the DIY toolbox, the paper also proposes a novel approach to store and sync the output configuration process on the cloud platform to make it available for use and possible customization. The cloud-based configuration server also has the ability to control the appliances and devices from outside of the local premises by just logging into server and triggering a few HTTP requests using the virtual objects interface.

Lastly, the work also proposes and enables the general public to add a generic entity called the smart space. The smart space can be perceived as any IoT world comprised of connected virtual resources aiming to do something in a more useful and automated way. The idea of this smart space is delineated by a smart home-based case study, which is implemented as an example scenario for the smart space, and numerous activities and snapshots of the case study are presented and discussed.

The rest of the paper is organized as follows: Section 2 covers major contributions in the related areas and provides an overview of similar research contributions. Section 3 outlines the conceptual architecture for the proposed work and briefly discusses the process to achieve the goal. Section 4 presents the detailed design of the system and provides the interaction model to describe the sequence of operations. Section 5 discusses the implementation environment and the technology stack used for the components used in the work. Section 6 covers the major steps of the execution results and provides snapshots of the proposed system. Section 7 gives an overview of the implementation of the smart home case study and discusses the ways to interact with the IoT server. Section 8 highlights the significance of the proposed work and compares it with previous works. Section 9, finally, concludes the paper and identifies future directions.

2. Related Work

The IoT vision is the realization of a worldwide network of connected and inter-operable smart devices that can provide services to people. Although the individual technologies in terms of communication and devices have improved tremendously, the implementation and realization of such a complex network are still in their nascent stage. The end goal is to have plug-n-play smart objects that can be deployed in any environment with an inter-operable interconnection backbone that allows them to blend with other smart objects around them. Standardization of frequency bands and protocols plays a pivotal role in accomplishing this goal. To provide a high-quality user experience, we need to find ways to overcome the hurdles, which include coping with the heterogeneity of the hardware and mass involvement of the general public in the IoT development and adaptation process. The first issue is being abstracted out with the help of a middleware-based solution and service-orientation. The second issue is of greater importance because in our view, the realization of a successful worldwide

IoT implementation is not possible without the involvement of the masses in the development of IoT, and this leads to the intervention of visual programming-based techniques; DIY, in other words.

Recently, there have been several efforts focused towards mass DIY in the fields of electronic device design, creation and programming. Some efforts are based on hardware boards, which come with electronic block modules for the general public to combine these blocks in any way they like and express their creativity by creating new and smarter things. Raspberry Pi [25] and Arduino [26] boards are some of the first and most popular efforts in this regard. These boards provide their own programming environment, and the users must be able to know programming languages such as Python or Java to write code for these boards. Microsoft .NET Gadgeteer [27] is another open-source toolkit for creating customized electronic devices by combining smaller electronic blocks onto a main board. The main board of the .NET Gadgeteer system has an embedded processor and sockets to connect simple plug-and-play Gadgeteer modules, which includes display, camera, networking, storage and sensors. The toolkit is based on the .NET programming language. NET Gadgeteer is aimed at exciting students about learning programming, electronics and design using an object-oriented environment of development.

SAM [28] is another Kickstarter [29] project, which provides blocked electronics modules, which can be used by inventors, creators, designers, and so forth, without any distinction of being a beginner or an expert in the field. The main theme of the SAM project is to combine hardware, software and the Internet. All the block modules in the SAM kit are wireless, and the Python language is used to program the modules. Not only is the DIY approach limited to basic hardware kits that can be combined as the user likes, but a recent trend is to investigate generic electronics with a higher degree of customizability and to involve the general public in the design and creation of such products, as has been investigated by Mazzei et al. [30]. The same idea is backed by many other studies such as Feki et al. [31], where DIY has been considered among the future trends in the field of IoT development. The study by Scott and Chin [32] provides an instance of the application of the DIY approach to IoT development based on low-cost Systems on Chip (SoC) as suggested in the previous paragraphs. A more recent application of the DIY IoT approach to digital agriculture in the form of crop growth monitoring and irrigation decision support, and so forth, is presented by Jayaraman et al. [33].

The Internet is a key player in the implementation and adaptation of IoT. The World Wide Web (WWW) and web services have also been used to provide a medium for makers or creators to share their inventions and creation. This way, they may be able to aggregate and reuse creations of other people to make more useful and smarter things. Pachube [34] is a web-centric service to aggregate streams of data “feeds” to acquire and store information related to different types of sensing devices and the data they produce over time. It also provides the capabilities of processing, integration and data visualization in the form of a collection of applications and is based on the idea of “triggers”. A trigger can be defined as the arrival of data from a resource (hardware or software), and in response, it can be forwarded to a specific URL based on some rule/condition or processed in order to activate some other triggers. In Pachube, the feeds’ integrations or triggers created by one person can be shared for use by others, enabling rapid development and creativity.

A more recent development came in the form of IBM’s Node-RED [35] with a focus to reducing the coding efforts and lowering the technical bar for the developers. Node-RED users wire together graphical nodes taken from a panel in order to create flows and then deploy these flows to get the results. The nodes in Node-RED represent devices, software platforms and web services [35]. The approach used by Node-RED is a better solution for enabling mass involvement in the realization of IoT, especially from a maker’s or creator’s perspective. Although these efforts have simplified the design and creation of things for the end-users, still, there is some level of experience required on the part of the developer. One way or the other, the developers must know how to program in order to use these hardware platforms or web services.

Glue.things [36] is another recent project that implements the concepts of device integration and real-time communication using the recent technologies of web sockets, MQTT and CoAP. The protocols

are utilized on real-time data stream networks to allow mashups of the data streams, add actions, etc. The final mashups are deployable in a distributed environment. The system especially focuses on the composition of data streams from web services and IoT devices with web interfaces. The main aim of Glue.things is to utilize web technologies for providing interoperability platform with REST APIs, JSON data models, web sockets, etc. The mashup interface is based on Node-RED. Another important aspect of the project is the utilization of well-established open source technologies.

Super Stream Collider (SSC) [37] is another platform that helps enable everyone, from novice IoT users to expert programmers, to develop IoT applications in the form of near-real-time data streams. The web-based interface for SSC enables anyone to create their own mashups by combining linked data sources and linked streams to create resources that can be used as applications for IoT scenarios. The system supports the drag-n-drop technique with a SPARQL/CQELS query language based editor. As the platform is intended for large data acquisitions through streams, it utilizes cloud infrastructure for fetching the data, processing and dissemination of data.

As part of the OpenIoT project, a visual development approach has been presented by Kefalakis et al. [38]. The visual development tools are intended to be used as an Integrated Development Environment (IDE) for the support of the IoT application development lifecycle. The tools presented are based on a semantic IoT architecture and claim to be a minimal programming environment for IoT application development. They use a node-based user interface theme to allow the user to model service graphs and then convert them into SPARQL queries.

The OPEL software platform [39] is designed considering the following requirements. First, the IoT platform should be programmable with an easy programming language. In particular, the programming language should have high productivity, portability and extendibility to enlarge the IoT ecosystem. A good candidate is the JavaScript (JS) language. Second, the platform should provide high-level APIs for easy application development. The APIs need to provide several functions, including sensor and device management, communication, etc. Third, the platform should support multiple IoT applications. User should be able to install multiple applications with different functions, which can be executed concurrently. Then, the user can exploit multiple services on a single device. Lastly, the software platform should enable the companion IoT device to communicate with its host device (e.g., Android smartphone) and to be controllable via its host device.

Other tools like ThingWorx [40], Particle.IO [41] and Dweet.IO [42], as outlined in [22], are also used extensively, but they have more of a Platform-As-A-Service (PAAS) nature and do not offer much flexibility to the general public. These tools, in addition to that, are not open source, and thus, the source code cannot be tweaked if any custom-made requirement needs to be incorporated.

All these technologies, to the best of our knowledge, either fully focus on the traditional desktop paradigm or the fully web-based paradigm. We believe that the strength of both paradigms can be integrated to design an IoT product that is closer to an ideal product.

3. Proposed DIY-Based IoT Cooperation Architecture and Composition System

The proposed IoT cooperation network as shown in Figure 1 mainly consists of the virtual and physical domain. The virtual IoT cooperation network provides a graphical interface for users to customize IoT applications in an intuitive way. The physical devices from the physical domain are represented as virtual objects that encapsulate the behaviors of the physical devices. These virtual objects are combined to generate service objects. Users utilize these service objects to customize the application process, which is further deployed to control the physical IoT network. The physical IoT cooperation network consists of various IoT sensors/actuators, and these devices receive and parse the deployment processing information from the virtual domain and then operate accordingly. For every device and composed service, an XML file is maintained, which stores the attributes and the URI of the server and the basic operation performed by the service. These XML files are stored on local storage initially as in other previous frameworks. In the next step, the XML data are ingested by the cloud-centric web platform and represented using web interfaces. The interfaces also provide better

understanding of the virtual devices and the composed services. Moreover, the interface enables the user to control the devices associated with the platform, thus mitigating the need to go on premises in order to perform some action on the device.

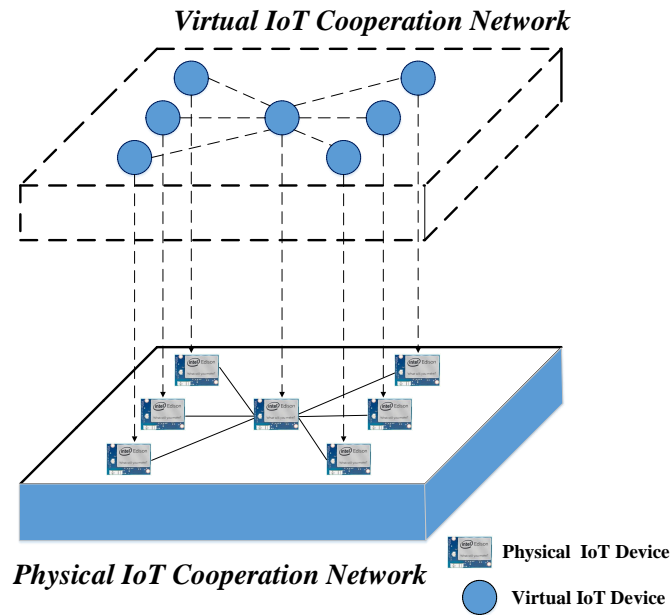


Figure 1. Mapping of cyber space to physical space.

Figure 2 shows the conceptual architecture of our proposed work. The DIY toolbox shows the black box representation of the framework, while the IoT node is the actual physical things capable of communication. IoT things can be communicated offline with the IoT toolbox or can be controlled using cloud-based web interfaces if the client is not on the same system in which the toolbox is installed. The cloud ingests XML configuration data from the toolbox and represents them using the web interface for a better user experience for the client application. The cloud also has the ability to add some more configurations, and these configurations can be pushed to the IoT toolbox.

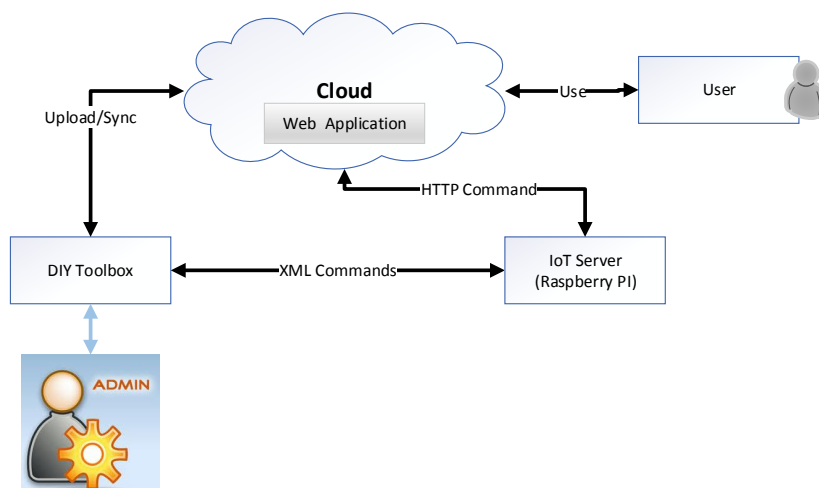


Figure 2. Conceptual architecture.

Figure 3 illustrates the white box representation of the DIY IoT toolbox conceptual view. The IoT toolbox is a layered architecture stacked together. Every layer has its own functionality and is decoupled from other layers, thus providing a modular approach.

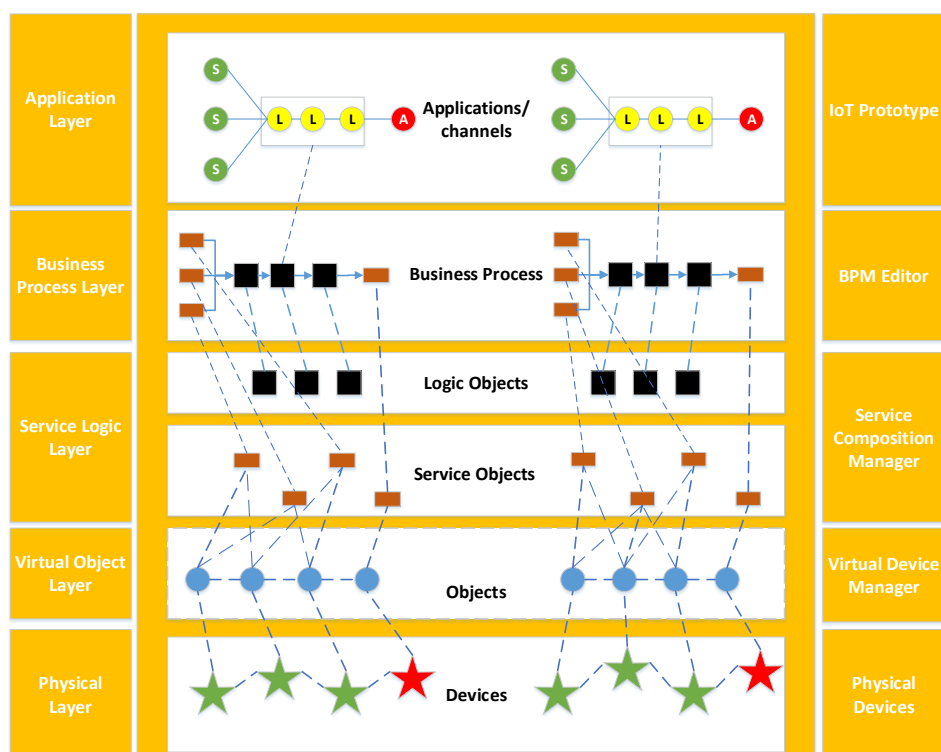


Figure 3. DIY toolbox layered architecture.

The physical cooperation network layer consists of IoT sensors and actuators; for instance, the thermistor, which acts as a resistor whose resistance varies significantly (more than in standard resistors) with temperature. This module's output approaches 5 V as the temperature increases. As the temperature decreases, it approaches 0 V. Other examples could be humidity sensors and temperature sensors. Actuators can be fans, LEDs or any other physical things capable of performing some action. The virtual object layer represents the physical things in cyber space. The behavior of these objects is the action they perform, and the properties of these objects are the attributes associated with them. Virtual objects interconnect with each other on the service logic layer. This layer is responsible for providing an intuitive and easy to use visual environment where the VOs obtained from Virtual Object Manager (VOM) are rendered in graphical form. The users can drag-drop these graphical objects to compose various services. Two kinds of services can be composed. Firstly, is with the association of sensors and actuators only without the intervention of any logic object. Secondly is that logic objects like fuzzy control can be added between sensors and the actuator to develop more smart IoT applications. Once the services are composed, they can be connected using the Business Process Modeling (BPM) editor in the businesses process layer, and real-time application requirement are designed on this layer. Finally, the application layer specifies the communication protocol and interfaces used for the physical IoT prototype. This layer consists of three types of node: the sensor node, the proxy node and the actuator node. The sensor node is responsible for collecting environmental data from physical IoT sensors. The actuator node takes charge of operating the physical IoT actuators. The proxy node is in charge of supporting communication between sensor nodes and actuator nodes. Logic objects provide intelligent prediction and control services to operate the IoT actuators. The communication channel defines the specified communication protocol for data and command transmission.

Figure 4 represents the block representation of the main components of the cloud-centric approach of storing the local XML configuration. The application is deployed on the Amazon Web Services (AWS) Compute Node which is also called EC2 node using Python and Flask. We believe that using EC2 is a better approach because most of the application logic has been performed by the DIY desktop

application, and the cloud only serves as a repository of XML files representing the already configured entity in a better UX interface. That being said, using a full-fledged cloud platform like AWS IoT would be considered overkill. We used the Python Untangle library [43] for efficient parsing of XML data and converting them to respective objects that can be used by the Python-based web applications. The XML connector provides interfaces to connect the configuration data with the cloud-based web application, allowing CRUD procedures to run on the applications.

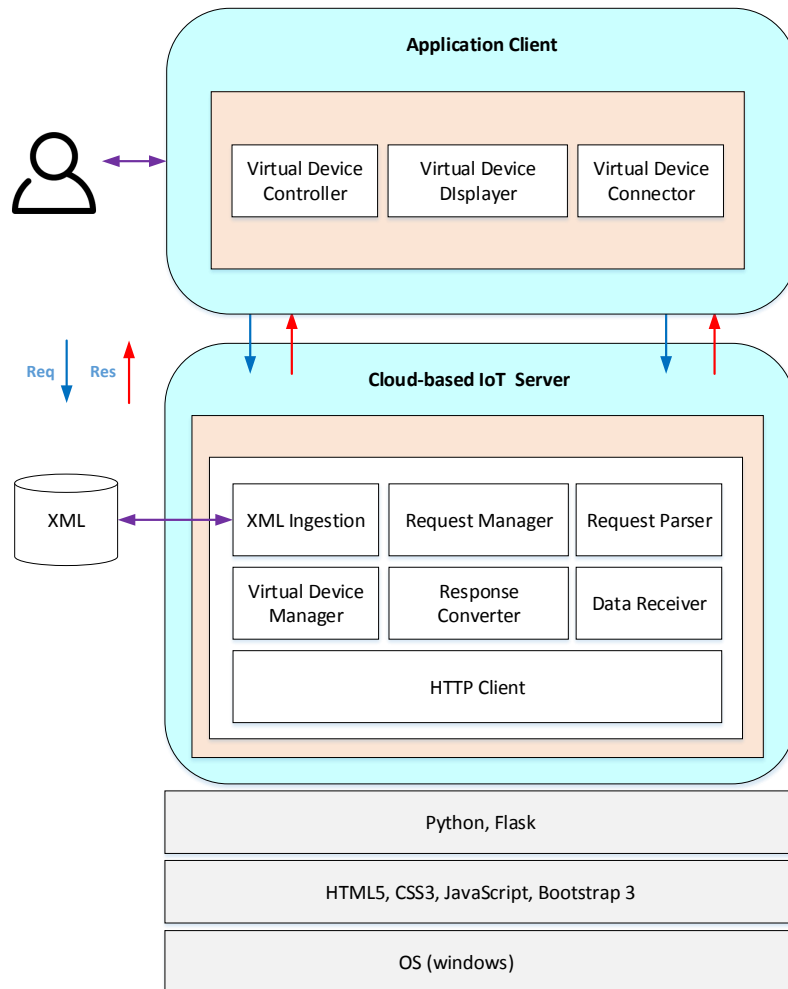


Figure 4. Cloud-centric application structure.

The virtual device manager provides functionality to manage the virtual devices. It is capable of creating the virtual devices, as well as connecting to the virtual devices. The virtual device manager provides a category that stores the virtual device information, and this category will be issued to the client whenever it receives the request from the app client. The request manager is responsible for accepting and processing the request from the application client. The request parser extracts the necessary parameters from the request and matches the related service to access the device. The HTTP client module is a separate communication module in a RESTful style, which is flexible and can be easily changed to adapt to different application protocols other than HTTP. The data receiver is in charge of receiving the response message from the IoT server. The response converter formats the response data, and after data formatting is completed, the processed data are visualized to the user on the web client. Moreover, for each virtual resource, there is an interface for controlling the state of the resource, and thus it could easily be controlled even if the user is not at his/her local premises.

4. Interaction Model

Figure 5 shows the sequence of connectivity among the DIY toolbox, the cloud-based IoT application and the physical devices.

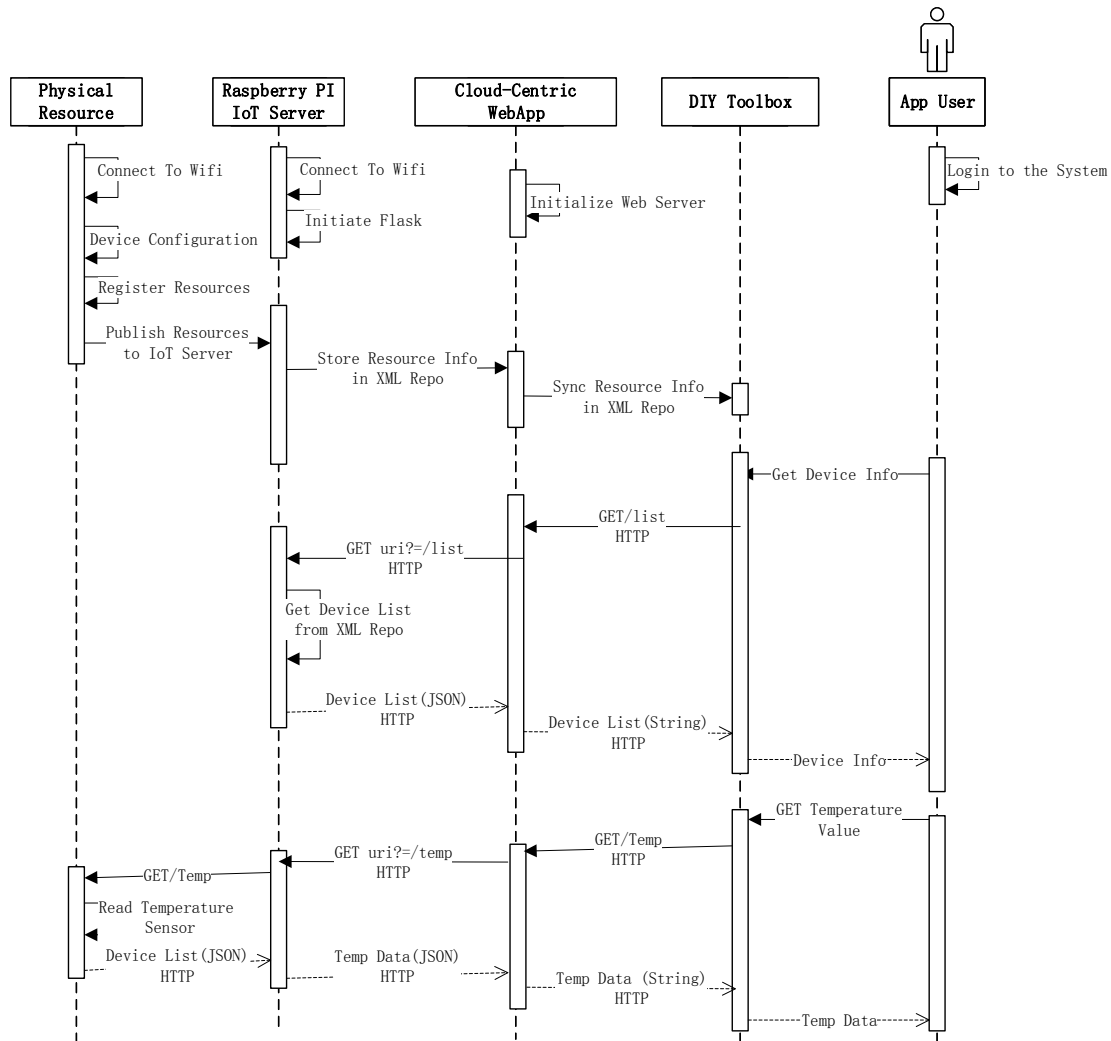


Figure 5. Sequence diagram of various operations within the proposed system.

It can be perceived from the interaction model diagram that on one end, there is an actual physical device, while on the other end, the app user entity resides. The devices are associated with the IoT server, which resides on the Raspberry PI. The Raspberry PI-based webserver registers the connected devices and assigns a unique URI, which can be published to the cloud-based web application. The cloud-based web application also initializes the web server using the Flask run method. The web server receives the resources' information in the form of XML and syncs it with the desktop-based IoT toolbox. Similarly, users can interact either with the DIY toolbox to get device information or interact with the cloud application if the user is not at his/her premises. Either way, the user is provided with the information. Therefore, in other words, if the user wants to get temperature from temperature sensor, the request will be made using either the DIY toolbox or the cloud-based IoT application, which will go all the way to the Raspberry PI-hosted IoT server. The server listens to the request and provides the data in XML format. The XML format data are converted into HTML form for rendering them in the web browser, and the end-user is informed about the operations and behavior of the sensors.

5. Implementation Details

In this section, an overview of the tools and technologies used in the implementation is presented and discussed. The project has three main components as shown in Figure 2. Therefore, the implementation tools have been summarized in three separate tables describing the technology stack for the respective component.

Table 1 illustrates the technology stack used for the IoT server hosted on Raspberry PI. The physical resources are associated with Raspberry PI and are implemented using the Python 3-based popular Model View Controller (MVC) framework known as Flask. Different LEDs, fan, temperature and humidity sensors have been used as physical resources and are registered in the Flask application. The application assigns a unique URI for every resource so that it can be uniquely identified by remote applications.

Table 1. Technology stack of the IoT server. IDE, Integrated Development Environment.

Component	Description
Hardware	Raspberry PI 3 Model B
Operating System	Raspbian
Memory	1 GB
Server	Flask Webservice
Resources	LED, Fan, Temperature Sensor, Humidity Sensor, Breadboard, Connecting Wires
Libraries	General Purpose Input/Output (GPIO), Untangle for XML Parsing, Jinja Template
IDE	Vim, PyCharm (Remote Access)
Programming Language	Python 3

Table 2 depicts the development environment of the cloud-based IoT configuration repository. The repository is deployed on the AWS EC2 compute node and is developed using the Python programming language. Flask, an MVC model, is used for better project organization and the need to bypass any middleware technologies. In order to present the information in a better way, we use Bootstrap 3, which is a responsive design-based framework. The client can interact with the cloud-based configuration repository platform and requests the Raspberry PI-based IoT server using the HTTP RESTful services. The server listens to the request on the designated port and performs the action on the associated physical resource.

Table 3 summarizes the technology stack used to implement the Do-It-Yourself (DIY) toolbox. The main functionality of the system is to provide a canvas where the general public can develop their application logic by just using the dragging and dropping functionality. The wonders of windows forms are utilized. In the future, we will focus on developing this using web-based technologies in order to keep all the modules in the same technologies, mitigating the need to learn multiple technologies for the use and possible improvement of the system.

Table 2. Technology stack of the cloud-based web application.

Component	Description
Operating System	Linux AWS EC2 Compute Node
IDE	Sublime Text 3, PyCharm
Programming Language	Python, Flask, JavaScript, HTML, CSS
Libraries and Framework	Untangle for Parsing XML, Bootstrap3, Jinja3 for Templating
Core Programming Language	Python 3
Browser	Chromium, Google Chrome, Firefox, Safari
Server	Flask-based Webservice
Persistence	XML Repository in Sync with the DIY Toolbox

Table 3. Technology stack of the DIY toolbox.

Component	Description
Operating System	Windows 8, 64 bits
CPU	Intel (R) Core(TM) i5-4570 CPU @3.20 GHz
Primary Memory	12 GB IDE
Visual Studio	2017, Community Version
Programming Language	C Sharp
Framework	.Net Framework 4.5
Libraries	Windows Form, KRBTABControl for implementing Control tabs
Persistence	XML Files in Sync with the Cloud App

6. Execution Result

The execution sequence of the proposed system is described in Figure 6. First off, the toolbox enables application users to design the logic of the application. The toolbox provides the abstract view of the virtual device and its behavior. These virtual devices and their behaviors can be used to compose services. Once the services are composed, the configuration is stored in XML format. These configuration files are ingested by the cloud-based web-app, and the same configuration is displayed using web interfaces for better User Experience (UX). The end-users can then either interact with local configuration files or the remote cloud web interfaces to send actual commands to the IoT server, which is hosted on Raspberry PI. The web server receives the request, parses it in the respective format and sends it to the designated address.

Figure 7 shows the overall sitemap and architecture diagram of the cloud-centric configuration repository web application. It takes sensors and actuator data in XML format files and represents them on web pages. Furthermore, as a result of the service composition in the DIY toolbox, the configuration settings are stored in the form of XML files. These XML files are synced on the cloud, and the web application parses these files to generate an interface that gives a better view of what is inside a particular service.

Figure 8 shows the windows form canvas where the user can drag and drop the virtual device to compose services. The arrows represent the connection between devices. The devices at both ends are the virtual representation of physical devices that are registered with the IoT server. Once the services are composed, the configuration settings are stored in XML format, as shown Figure 8b. These are the XML files that are ingested by the cloud-based web application. The cloud-based web application consumes these files and represents the same information in a clearer and more robust way.

Figure 9 shows the equivalent virtual device representation of the XML data being ingested. Every device has some attributes like the location of the device, the communication protocol, URI and the properties.

The devices are based on XML files that are pulled from the DIY IoT toolbox. It also has the ability to add new a virtual device using the web form shown in Figure 10.

Once the form is submitted, the device info is stored in the form of XML and pushed to the DIY IoT toolbox, thus keeping both the offline and cloud platform in sync. The service composition is performed on the service composition layer using simple drag and drop, as shown in the figure. Once the services are composed, the resultant XML files are pulled from the toolbox repository and represented using the web interface as shown in Figure 11.

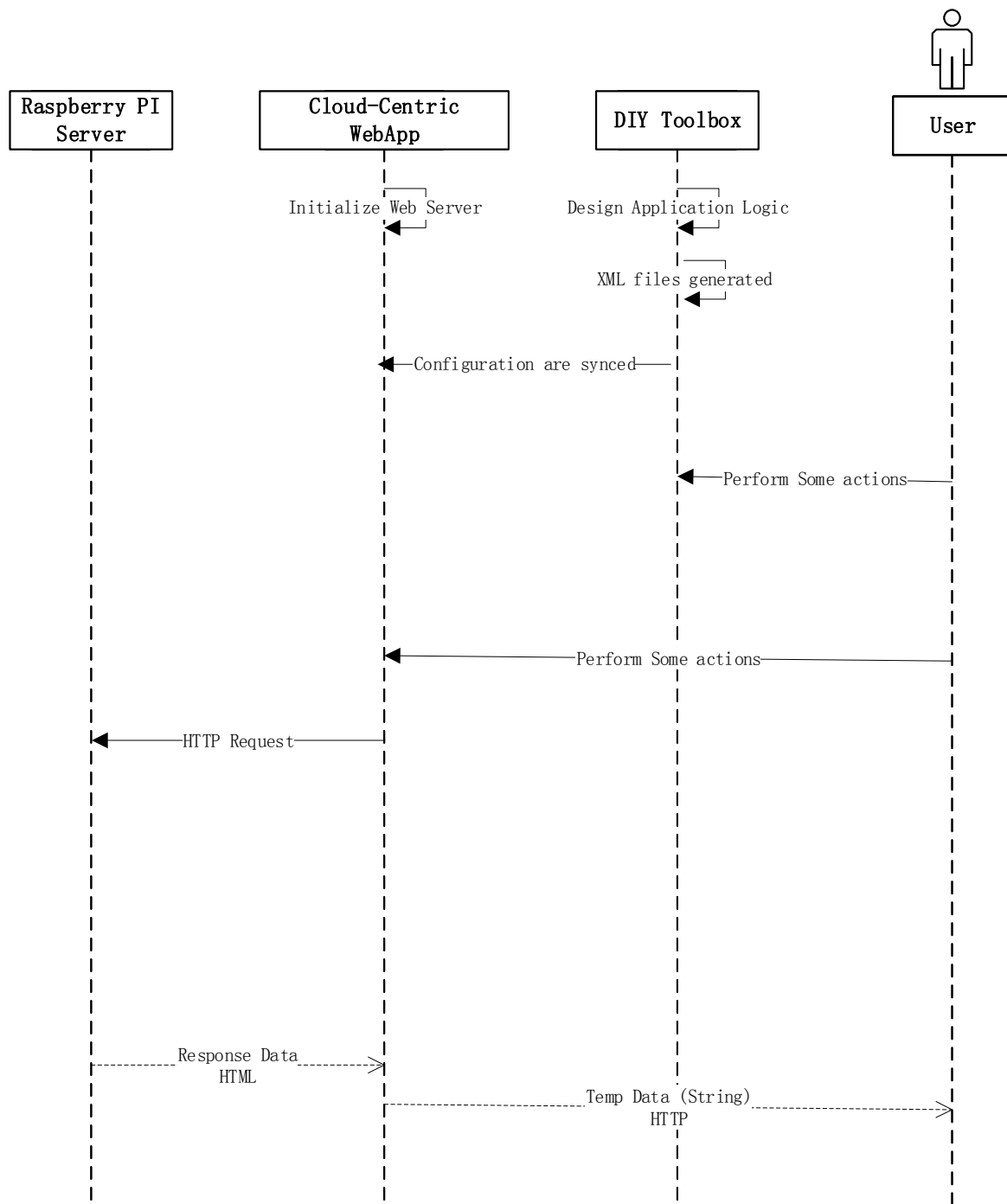


Figure 6. Execution sequence.

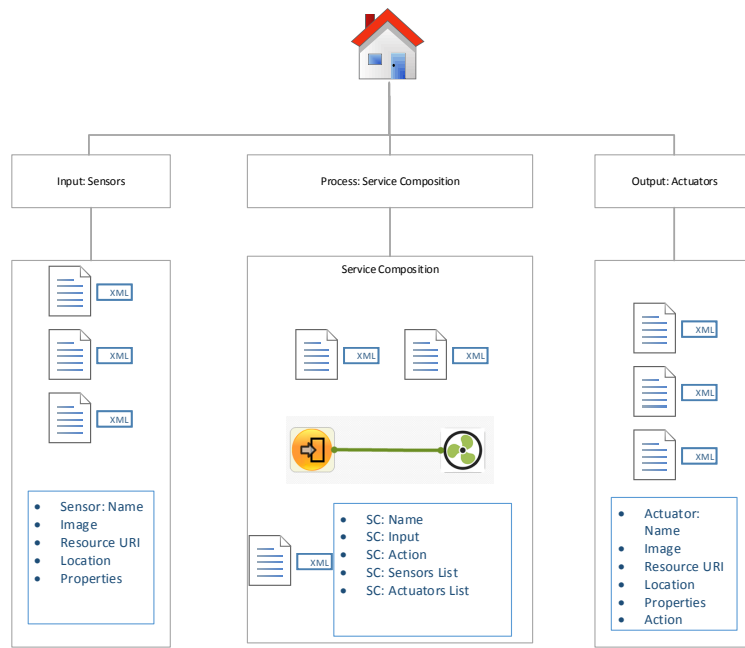


Figure 7. Cloud-based web application site map.

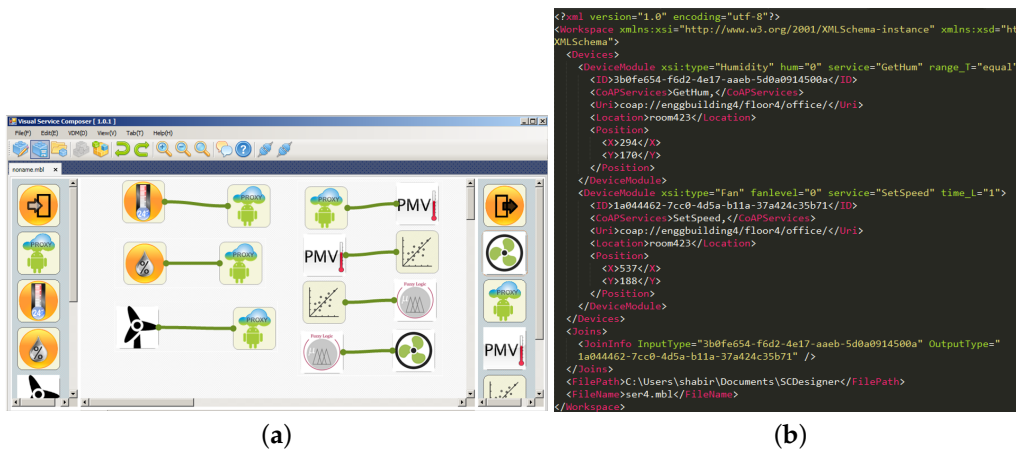


Figure 8. DIY toolbox service composition. (a) Virtual device; (b) XML representation.

Optimization of Interfaces of IoTCooperation Framework for better User Experience

Input Data Virtual Nodes Physical Devices Services IFM Objects

Virtual Devices

No	Location	Type	CoAP URI	Properties
1	Illumination	LightSensor	illuminationurl	sd
2	room423	HumidSensor	coap://192.168.0.11:5681/humidty	GetHum
3	room423	WindSensor	coap://192.168.0.11:5681/wind	GetWind
4	room423	Proxy	coap://192.168.0.11:5681/proxy	GetProxy
5	room423	TempSensor	coap://192.168.0.11:5681/temp	GetTempC
6	room423	Fan	coap://192.168.0.11:5681/fan	SetFan

Figure 9. Cont.


```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Devices>
3   <Device>
4     <Uri>illumination.uri</Uri>
5     <Type>LightSensor</Type>
6     <Location>Illumination</Location>
7     <Properties>
8       <P>sd</P>
9     </Properties>
10  </Device>
11  <Device>
12    <Uri>coap://192.168.0.15:5683/humidity</Uri>
13    <Type>HumidSensor</Type>
14    <Location>room423</Location>
15    <Properties>
16      <P>GetHum</P>
17    </Properties>
18  </Device>
19  <Device>
20    <Uri>coap://192.168.0.13:5683/wind</Uri>
21    <Type>WindSensor</Type>
22    <Location>room423</Location>
23    <Properties>
24      <P>GetWind</P>
25    </Properties>
26  </Device>
27  <Device>
28    <Uri>coap://192.168.0.5:5683/proxy</Uri>
29    <Type>Proxy</Type>
30    <Location>room423</Location>
31    <Properties>
32      <P>GetProxy</P>
33    </Properties>
34  </Device>
35  <Device>
36    <Uri>coap://192.168.0.21:5683/temp</Uri>
37    <Type>TempSensor</Type>
38    <Location>room423</Location>
39    <Properties>
40      <P>GetTemp</P>
41    </Properties>
42  </Device>
43  <Device>
44    <Uri>coap://192.168.0.12:5683/fan</Uri>
45    <Type>Fan</Type>
46    <Location>room423</Location>
47    <Properties>
48      <P>SetFan</P>
49    </Properties>
50  </Device>
51 </Devices>

```

(b)

Figure 9. Virtual device representation. (a) Virtual device web interface; (b) XML representation.



Optimization of Interfaces of IoT Cooperation Framework for better User Experience

[Ingest Data](#) [Virtual Nodes](#) [Physical Devices](#) [Services](#) [BPM Objects](#)

Add Device

Device Name

Image File No file chosen

Device URI


Device Type

Location

Device Methods

Device Property

Figure 10. Interface for adding a device.



Optimization of Interfaces of IoTCooperation Framework for better User Experience

Ingest Data Virtual Nodes Physical Devices Services BPM Objects

No	Input	Output
1	Device ID: e1b0a155-0029-4c65-8234-ed6a2d1b1f31 Services: GetTempC, GetTempF, URI: coap://coputer-engg/office/ Location: room423 Position: 275, 164	Device ID: 5c49cc3f-76c2-4ac6-80f4-942cf8a1cf08 Services: SetSpeed, URI: coap://enggblding4/floor4/office/ Location: room423 Position: 560, 177
2	Device ID: 3235937a-5370-4f43-9c8f-344d7ba61879 Services: GetWind, URI: coap://enggblding4/floor4/office/ Location: room423 Position: 286, 186	Device ID: d0f74334-21b7-41ef-8b95-262021085ec4 Services: GetPMV, URI: coap://enggblding4/floor4/office/ Location: room423 Position: 540, 193
3	Device ID: 4052a336-578f-4a7e-b2f9-716097aa8644 Services: GetHum, URI: coap://enggblding4/floor4/office/ Location: room423 Position: 289, 164	Device ID: ef3a2db9-9d87-432d-9e33-8ecae1924691 Services: SetSpeed, URI: coap://enggblding4/floor4/office/ Location: room423 Position: 499, 168
4	Device ID: 3b0fe654-f6d2-4e17-aaeb-5d0a0914500a Services: GetHum, URI: coap://enggblding4/floor4/office/ Location: room423 Position: 294, 170	Device ID: 1a044462-7cc0-4d5a-b11a-37a424c35b71 Services: SetSpeed, URI: coap://enggblding4/floor4/office/ Location: room423 Position: 537, 188

Figure 11. Services representation in the cloud.

7. Case Study: Smart Home

In order to evaluate the effectiveness of the proposed work, a smart home case study has been implemented as part of the experimental work. The case study also describes the way communication occurs between the cloud, IoT nodes and the DIY toolbox. Figure 12 shows the implementation environment for the case study presented in the subsequent section. The Raspberry PI 3, which acts as the IoT server, has some physical resources connected. For the smart home case study, the fan actuator, temperature sensor and light LEDs are used, which are clearly visible in the figure.

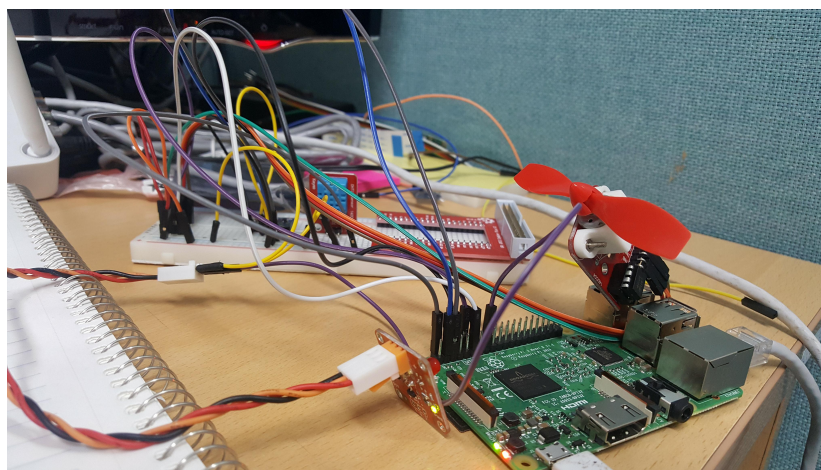


Figure 12. Implementation environment.

For implementation purposes, we define a generic entity, which can act as a wrapper for the smart world. The entity is named the smart space. Smart homes, smart cities and smart farms can be added as example contents.

Add Smart Space

The first step is to add a smart space to the web-based application. The smart space is a generalized content type for the smart world. The smart space could be anything like a smart home, a smart farm,

etc. For the case study, the smart home is added using the web form shown in Figure 13. The name attribute of the form is the name of the smart space, for instance, the smart home in this case. The form, additionally, allows users to upload an image of their choice for the smart space. Moreover, a drop down list of virtual resources appears from which the resources can be added to the smart space. In Figure 13, the name is assigned as “smart home”, and the lights, fan and temperature and humidity sensors are added as virtual resources.

Figure 13. Add smart space interface.

Once the smart home is added, the system presents a pictorial view of the smart home with various virtual resources added during the process described before. The resources are linked with the actual physical resources, and clicking on any of them fires a request to the IoT server. The IoT server, which is hosted on Raspberry PI, listens to the request and performs the action on the respective device.

Table 4 summarizes numerous endpoints of communication between the cloud-based IoT application and the Raspberry PI-based IoT server.

Table 4. Communication endpoints.

URL	Action
http://rasp-localhost/fan?op=on&dir=clk	Turn on the fan in the Clockwise Direction
http://rasp-localhost/fan?op=on&dir=anticlk	Turn on the fan in the Counter-Clockwise Direction
http://rasp-localhost/fan?op=off	Turn Off the Fan
http://rasp-localhost/led?op=on	Turn On the LED Light
http://rasp-localhost/led?op=off	Turn Off the LED Light
http://rasp-localhost/get-sensor-data	Get the Current Temperature from the Temperature Sensor

Figure 14 shows some snapshots of various interfaces based on the communication endpoints described in Table 4, and their respective actions are displayed in the dialog box.

Figure 14a shows the overview of the smart space. The pictorial representation of the virtual resources is presented for various locations of the home. Figure 14b illustrates the process of controlling the LED lights. The LED bulb is clicked to display a modal dialog showing the current status of the LED and the option to change the status. In the same way, in Figure 14c, the dialog box that pops up as a result of clicking on the fan icon is presented. The fan can be turned on in the clockwise direction or in the counter-clockwise direction. If the current status is turned on, a button will appear instead, which will turn off the fan. Lastly, in Figure 14d, the temperature sensor icon is clicked to get the current room temperature from the sensor.



Figure 14. Smart home snapshots. (a) Smart home initial view; (b) Light control; (c) Fan control; (d) Temperature reading.

8. Comparison and Significance

As discussed in the earlier section of the paper, to the best of the authors' knowledge, this work is a novel attempt towards the cloud-based DIY toolbox. Many similar applications exist, but they are either fully web based or fully desktop based. In the proposed work, the compute node service of the cloud platform is utilized in order to boost the performance of the web application, and the specialized cloud IoT is not considered, because the authors believe it to be overkill in this scenario. In order to assess the effectiveness of the proposed system, a comparative analysis of the system with the related tools discussed in the Related Work Section has been carried out. The result has been summarized in Table 5. The features that play a vital role for any application are considered for this study. It has been shown in the table that the proposed system outperforms the related tools in many aspects. It can be seen from the table that Node-Red is a somewhat similar approach. The main difference is that Node-Red renders blocks of code visually and combines these blocks to form flow of the program, which means the DIY support is partial and to some extent technical expertise is required. Another application named SAM also performs similarly. The main problem is that the program is inherently written for a specific platform and thus will not work if the underlying platform changes. Moreover, the DIY toolboxes, like OPEL Software Platform and Super Stream Collider, use web-based APIs to semantically model the behavior of the IoT application, but these applications cannot be accessed offline. Similarly, Particle.IO and Dweet.IO are platforms as a service and fully cloud based, so they do not give much flexibility to the general public.

Table 5. Comparative analysis of the proposed system with the related tools.

No	Name	Open-Source	DIY Support	Remote Access	Configuration Repository	Offline Access	Cloud Support	Application Type	Generic Entity Support	Programming Language
1	SAM	No	Partial	No	XML	Yes	Yes	Desktop	No	JavaScript/Python
2	Super Stream Collider	No	Partial	Yes	JSON/PLSQL	No	No	Web	No	JavaScript
3	OPEL	Yes	No	Yes	JSON	No	No	Mobile/Web	No	JavaScript
4	Node-red	Yes	Partial	Yes	JSON	No	No	Web	No	Node.js
5	Glue.thing	No	Yes	Yes	JSON	No	No	Web	No	Node.js
6	Thing.work									
7	OpenIoT	Yes	No	Yes	XML	No	Yes	Web	No	Java
8	Particle.IO	No	No	Yes	XML	No	Yes	Middleware	Yes	Java
9	Dweet.IO	Yes	No	No	JSON	No	No	Middleware	No	Multi-language Support
10	Proposed System	Yes	Yes	Yes	XML	Yes	Yes	Hybrid (Web and Desktop)	Yes	Python

In this work, the concept of the generic entity, known as the smart space, is introduced, and a smart home-based case study has been implemented as a proof of concept. The generic entity can be extended to scenarios like smart cities and smart farms, which also supports the significance of the work. Moreover, the demands to have an application that can work offline using traditional desktop applications, that can work online from remote places and that supports different protocols are increasing exponentially, and this paper is the first attempt to address all of the above-mentioned issues.

9. Conclusions

This paper outlines the procedure for the design and implementation of a novel architecture for enabling the general public to build IoT applications with zero coding. The configuration settings are stored in XML format and are ingested by the cloud-based web application for better UX and remote connectivity. For the IoT server, Raspberry PI is used, and the powerful API of Flask MVC has been leveraged. Resources are represented in XML format and are kept in sync between the DIY toolbox and the cloud-based web application. End-users can either use the toolbox on premises or use cloud-based web application to remotely interact with the smart space. The novelty of the presented work is the distribution of configuration settings both in local storage and cloud storage, thus providing users with the flexibility to remotely access the application even if the users are not at their premises. A smart home case study has been implemented for the evaluation of the significance of the proposed work, and the various interfaces have been discussed in detail. The significance of this work has been outlined with a benchmark study of the proposed system with similar tools, and it has been shown that the system performs better with respect to the existing tools. The future direction of this project can be to allow users to model and design the application logic using web-based technologies like HTML5 canvas and drawing tools, thus making both systems imitate the majority of their functions.

Acknowledgments: This work was supported by Institute for Information and communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00756, Development of interoperability and management technology of IoT system with heterogeneous ID mechanism), and this research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2016-0-00313) supervised by the IITP (Institute for Information and communications Technology Promotion), Any correspondence related to this paper should be addressed to Do Hyeun Kim.

Author Contributions: Shabir Ahmad conceived the idea for this paper, designed the experiments and wrote the paper; Lei Hang assisted in model designing and experiments. Do Hyeun Kim conceived the overall idea of Cloud-Centric approach for DIY IoT, and proof-read the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cloutier, G.; Papin, M.; Bizier, C. Do-it-yourself (DIY) adaptation: Civic initiatives as drivers to address climate change at the urban scale. *Cities* **2018**, doi:10.1016/j.cities.2017.12.018.
2. Valderrama, C.; Vachaudes, J.; Bettens, F.; Vinci dos Santos, F.; Menezes, N.; Roelands, M. The DiY Smart Experiences Project: A European Endeavour Removing Barriers for User-generated Internet of Things Applications. *Archit. Internet Things* **2011**, doi:10.1007/978-3-642-19157-2_11.
3. Coetzee, L.; Eksteen, J. The Internet of Things-promise for the future? An introduction. In Proceedings of the IST-Africa Conference Proceedings, Gaborone, Botswana, 11–13 May 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 1–9.
4. Peña-López, I. *ITU Internet Report 2005: The Internet of Things*; ITU Telecom World: Hong Kong, China, 2005.
5. IBM. The Ideas That Shaped Centuries. Available online: <http://www-03.ibm.com/ibm/history/ibm100/us/en/ideas/sep2011china.html> (accessed on 29 November 2017).
6. Atzori, L.; Iera, A.; Morabito, G. The internet of things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805.
7. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660.

8. Serpanos, D.; Wolf, M. Industrial Internet of Things. In *Internet-of-Things (IoT) Systems*; Springer: Berlin, Germany, 2018; pp. 37–54.
9. Fenn, J.; LeHong, H. *Hype Cycle for Emerging Technologies, 2011*; Gartner: Stamford, CT, USA, 2011.
10. Miorandi, D.; Sicari, S.; De Pellegrini, F.; Chlamtac, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Netw.* **2012**, *10*, 1497–1516.
11. López-de Ipiña, J.B.D.; Moya, F. *Ubiquitous Computing and Ambient Intelligence*; Springer: Berlin, Germany, 2017.
12. Kelly, S.D.T.; Suryadevara, N.K.; Mukhopadhyay, S.C. Towards the implementation of IoT for environmental condition monitoring in homes. *IEEE Sens. J.* **2013**, *13*, 3846–3853.
13. Gama, K.; Touseau, L.; Donsez, D. Combining heterogeneous service technologies for building an Internet of Things middleware. *Comput. Commun.* **2012**, *35*, 405–417.
14. Atzori, L.; Iera, A.; Morabito, G. From “smart objects” to “social objects”: The next evolutionary step of the internet of things. *IEEE Commun. Mag.* **2014**, *52*, 97–105.
15. Xiao, G.; Guo, J.; Da Xu, L.; Gong, Z. User interoperability with heterogeneous IoT devices through transformation. *IEEE Trans. Ind. Inform.* **2014**, *10*, 1486–1496.
16. Anderson, C. *Makers: The New Industrial Revolution (2013) Crown Business*; Crown Business: New York, NY, USA, 2012.
17. Bailey, C. *Making Is Connecting: The Social Meaning of Creativity From DIY and Knitting to YouTube and Web 2.0*; Polity Press: Cambridge, UK, 2013.
18. Leadbeater, C.; Miller, P. *The Pro-Am Revolution: How Enthusiasts are Changing Our Society and Economy: How Enthusiasts Are Changing Our Society and Economy*; Demos: London, UK, 2004.
19. Salamone, F.; Belussi, L.; Danza, L.; Ghellere, M.; Meroni, I. How to control the Indoor Environmental Quality through the use of the Do-It-Yourself approach and new pervasive technologies. *Energy Procedia* **2017**, *140*, 351–360.
20. De Roeck, D.; Slegers, K.; Criel, J.; Godon, M.; Claeys, L.; Kilpi, K.; Jacobs, A. I would DiYSE for it! A manifesto for do-it-yourself internet-of-things creation. In Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design, Copenhagen, Denmark, 14–17 October 2012; ACM: New York, NY, USA, 2012; pp. 170–179.
21. Eddy, N. Gartner: 21 Billion IoT Devices To Invade By 2020—InformationWeek. Available online: <http://www.informationweek.com/mobile/mobile-devices/gartner-21-billion-iot-devices-to-invade-by-2020/d/d-id/1323081> (accessed on 23 January 2018).
22. Vestergaard, L.S.; Fernandes, J.; Presser, M. Creative coding within the Internet of Things. In Proceedings of the Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.
23. Shelby, Z.; Hartke, K.; Bormann, C. *The Constrained Application Protocol (CoAP)*; Internet Engineering Task Force (IETF): Fremont, CA, USA, 2014.
24. Org, E. Eclipse Californium. Available online: <http://www.eclipse.org/californium/> (accessed on 30 November 2017).
25. PI, R. What is Raspberry PI? Available online: <https://www.raspberrypi.org/help/what-is-a-raspberry-p/> (accessed on 30 November 2017).
26. Rduino. Rduino. Available online: <http://www.arduino.cc/> (accessed on 30 November 2017).
27. Villar, N.; Scott, J.; Hodges, S.; Hammil, K.; Miller, C. .NET gadgeteer: A platform for custom devices. In Proceedings of the International Conference on Pervasive Computing, Lugano, Switzerland, 19–23 March 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 216–233.
28. SAM: The Ultimate Internet Connected Electronics Kit. Available online: <https://www.kickstarter.com/projects/1842650056/sam-the-ultimate-internet-connected-electronics-ki> (accessed on 1 December 2017).
29. Kickstarter. Available online: <https://www.kickstarter.com/> (accessed on 1 December 2017).
30. Mazzei, D.; Montelisciani, G.; Fantoni, G.; Baldi, G. Internet of Things for designing smart objects. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Korea, 6–8 March 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 293–297.
31. Feki, M.A.; Kawsar, F.; Boussard, M.; Trappeniers, L. The internet of things: The next technological revolution. *Computer* **2013**, *46*, 24–25.

32. Scott, G.; Chin, J. A DIY approach to pervasive computing for the Internet of things: A smart alarm clock. In Proceedings of the 2013 5th Computer Science and Electronic Engineering Conference (CEEC), Colchester, UK, 17–18 September 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 57–60.
33. Jayaraman, P.P.; Palmer, D.; Zaslavsky, A.; Georgakopoulos, D. Do-it-Yourself Digital Agriculture applications with semantically enhanced IoT platform. In Proceedings of the 2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, 7–9 April 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 1–6.
34. Pachube—The Internet of Things Real-Time web service and applications. Available online: <http://www.appropedia.org/Pachube> (accessed on 2 December 2017).
35. Heath, N. How IBM's Node-RED is hacking together the internet of things. Available online: <http://www.techrepublic.com/article/node-red/> (accessed on 3 December 2017).
36. Kleinfeld, R.; Steglich, S.; Radziwonowicz, L.; Doukas, C. Glue. things: A Mashup Platform for wiring the Internet of Things with the Internet of Services. In Proceedings of the 5th International Workshop on Web of Things, Cambridge, MA, USA, 8 October 2014; ACM: New York, NY, USA, 2014; pp. 16–21.
37. Quoc, H.N.M.; Serrano, M.; Le-Phuoc, D.; Hauswirth, M. Super stream collider-linked stream mashups for everyone. In Proceedings of the Semantic Web Challenge Co-Located with ISWC2012, Boston, MA, USA, 11–15 November 2012.
38. Kefalakis, N.; Soldatos, J.; Anagnostopoulos, A.; Dimitropoulos, P. A visual paradigm for IoT solutions development. In *Interoperability and Open-Source Solutions for the Internet of Things*; Springer: Berlin, Germany, 2015; pp. 26–45.
39. Lee, H.; Sin, D.; Park, E.; Hwang, I.; Hong, G.; Shin, D. Open software platform for companion IoT devices. In Proceedings of the 2017 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 8–10 January 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 394–395.
40. Enterprise IoT Solutions and Platform Technology. Available online: <https://www.thingworx.com/> (accessed on 23 January 2018).
41. Particle: Connect Your Internet of Things (IoT) Devices. Available online: <https://www.particle.io/> (accessed on 23 January 2018).
42. dweet.io—Share Your Thing- Like It Ain'T no Thang. Available online: <http://dweet.io/> (accessed on 23 January 2018).
43. XML Parsing Techniques. Available online: <http://docs.python-guide.org/en/latest/scenarios/XML/> (accessed on 6 December 2017).



© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).