



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

A Thesis

For the Degree of Master of Science

# IoT Cooperation Architecture based on OCF IoTivity and CoAP Protocol

Lei Hang

Department of Computer Engineering

Graduate School

Jeju National University

June 2017

*Dedicated to my families for being a constant source  
of support and encouragement!*

# Acknowledgment

First of all, I would like to express my gratitude to all those who helped me during the writing of this thesis. I gratefully acknowledge the help of my supervisor, Prof. Do-Hyeun Kim, who has offered me valuable suggestions in the academic studies. In the preparation of this thesis, he has spent much time reading through each draft and provided me with inspiring advice. Without his patient instruction, insightful criticism and expert guidance, the completion of this thesis would not have been possible.

Second, I would like to thank my thesis evaluation committee for their insightful comments and valuable suggestions during the thesis defense. Their input helped me in elevating the quality of this thesis.

Last, I really appreciate the support and encouragement given to my current lab mates Wenquan Jin, Songai Xuan, Israr Ullah and Muhammad Fayaz for their kind support and help during this endeavor.

To all of you, a heartfelt thanks for everything you did!

**Lei Hang**

# Table of Contents

<b>Abstract.....</b>	<b>1</b>
<b>1. Introduction.....</b>	<b>3</b>
<b>2. Related Work.....</b>	<b>8</b>
<b>2.1 IoT Composition Platform .....</b>	<b>8</b>
<b>2.2 IoT Standards .....</b>	<b>13</b>
<b>2.3 IoT Open Source Hardware.....</b>	<b>17</b>
<b>2.4 IoT Protocol.....</b>	<b>20</b>
<b>2.5 Open Source Machine Learning Software.....</b>	<b>23</b>
<b>3. Proposed IoT Cooperation Architecture and Composition System .....</b>	<b>26</b>
<b>3.1 Physical Cooperation Network Layer .....</b>	<b>30</b>
<b>3.2 Virtual Object Layer.....</b>	<b>34</b>
<b>3.3 Service Logic Layer.....</b>	<b>36</b>
<b>3.4 Business Process Layer .....</b>	<b>40</b>
<b>3.5 Application Layer.....</b>	<b>46</b>
<b>3.6 PMV (Predicted Mean Vote).....</b>	<b>48</b>
<b>3.7 PMV Prediction based on Linear Regression Algorithm .....</b>	<b>51</b>
<b>3.8 Fan Control based on Fuzzy Logic with PMV.....</b>	<b>54</b>
<b>4. Implementation .....</b>	<b>59</b>
<b>4.1 Virtual Device Manager .....</b>	<b>59</b>

<b>4.2</b>	<b>Service Composition Manager .....</b>	<b>65</b>
<b>4.3</b>	<b>BPM Editor.....</b>	<b>70</b>
<b>4.4</b>	<b>IoT Smart Space Prototype .....</b>	<b>76</b>
<b>4.5</b>	<b>PMV Calculation.....</b>	<b>84</b>
<b>4.6</b>	<b>PMV Prediction based on Linear Regression.....</b>	<b>85</b>
<b>4.7</b>	<b>Control based on Fuzzy Logic with PMV .....</b>	<b>88</b>
<b>5.</b>	<b>Experiment and Evaluation .....</b>	<b>92</b>
<b>6.</b>	<b>Conclusion.....</b>	<b>101</b>
	<b>References.....</b>	<b>102</b>

# List of Figures

<b>Figure 1: Comparison of the existing and proposed IoT cooperation system architecture ..</b>	<b>6</b>
<b>Figure 2: Generic architecture of the Glue.Things project.....</b>	<b>9</b>
<b>Figure 3: Conceptual architecture of IoT MAP .....</b>	<b>10</b>
<b>Figure 4: High-level architecture of the IoTLink.....</b>	<b>11</b>
<b>Figure 5: Layered architecture of the SSC Platform.....</b>	<b>11</b>
<b>Figure 6: OCF IoTivity architecture .....</b>	<b>14</b>
<b>Figure 7: oneM2M reference architecture.....</b>	<b>16</b>
<b>Figure 8: Arduino UNO.....</b>	<b>18</b>
<b>Figure 9: IoT cooperation network conceptual .....</b>	<b>26</b>
<b>Figure 10: Connection between Virtual Domain and Physical Domain.....</b>	<b>27</b>
<b>Figure 11: Sequence of the IoT Architecture based on IoT Proxy .....</b>	<b>29</b>
<b>Figure 12: Proposed IoT Composition Architecture .....</b>	<b>30</b>
<b>Figure 13: TinkerKit Thermistor Module .....</b>	<b>31</b>
<b>Figure 14: Humidity Sensor HIH-4030 .....</b>	<b>32</b>
<b>Figure 15: Wind Speed Sensor (SKU:SEN0170).....</b>	<b>33</b>
<b>Figure 16: Arduino L9110 Fan Module .....</b>	<b>33</b>
<b>Figure 17: Physical IoT Cooperation Network Structure .....</b>	<b>34</b>
<b>Figure 18: Virtual Device Manager Operation Configuration .....</b>	<b>36</b>
<b>Figure 19: Service Composition Manager Operation Configuration.....</b>	<b>38</b>
<b>Figure 20: BPM Editor Operational Configuration .....</b>	<b>41</b>
<b>Figure 21: BPM Deployment Engine Operation Configuration.....</b>	<b>42</b>
<b>Figure 22: IoT Application Development Procedure in terms of the BPL Perspective.....</b>	<b>44</b>
<b>Figure 23: IoT Application Prototype Conceptual .....</b>	<b>46</b>

<b>Figure 24: Design of the IoT Application Prototype.....</b>	<b>47</b>
<b>Figure 25: Fuzzy Sets for the Input Variable PMV .....</b>	<b>54</b>
<b>Figure 26: Fuzzy Sets for the Input Variable <math>\Delta</math>PMV.....</b>	<b>54</b>
<b>Figure 27: Fuzzy Sets for the Output Variable Fan Speed .....</b>	<b>55</b>
<b>Figure 28: Fuzzy Controller Block Diagram .....</b>	<b>57</b>
<b>Figure 29: IoT Smart Space Implementation in the Proposed Architecture.....</b>	<b>60</b>
<b>Figure 30: Virtual Device Manager Main Interface .....</b>	<b>61</b>
<b>Figure 31: Device Interface for Creating Virtual Objects.....</b>	<b>62</b>
<b>Figure 32: XML Representation of Created Virtual Devices .....</b>	<b>64</b>
<b>Figure 33: Service Composition Manager Interface.....</b>	<b>66</b>
<b>Figure 34: Service Object Process in Service Composition Manager .....</b>	<b>67</b>
<b>Figure 35: XML Representation of Service Objects.....</b>	<b>69</b>
<b>Figure 36: BPM Editor Interface .....</b>	<b>70</b>
<b>Figure 37: Loading Service Objects at BPM Editor .....</b>	<b>71</b>
<b>Figure 38: Generated BPMN in BPM Editor .....</b>	<b>72</b>
<b>Figure 39: Temperature Sensor XML Representation of BPMN.....</b>	<b>73</b>
<b>Figure 40: PMV Index XML Representation of BPMN.....</b>	<b>73</b>
<b>Figure 41: LR Predictor XML Representation of BPMN .....</b>	<b>74</b>
<b>Figure 42: Fuzzy Controller XML Representation of BPMN.....</b>	<b>74</b>
<b>Figure 43: Temperature Sensor XML Representation of BPMN.....</b>	<b>74</b>
<b>Figure 44: IoT Smart Space Application Prototype Structure .....</b>	<b>77</b>
<b>Figure 45: Implement Environment of Temperature Sensor .....</b>	<b>78</b>
<b>Figure 46: Implement Environment of Wind Speed Sensor .....</b>	<b>79</b>
<b>Figure 47: Implement Environment of Humidity Sensor.....</b>	<b>79</b>
<b>Figure 48: Implement Environment of Fan Actuator .....</b>	<b>80</b>



<b>Figure 49: Implement Environment of IoT Proxy .....</b>	<b>81</b>
<b>Figure 50: PMV Equation Parameter Definition .....</b>	<b>82</b>
<b>Figure 51: PMV Equation Implementation.....</b>	<b>83</b>
<b>Figure 52: PMV Execution Result.....</b>	<b>84</b>
<b>Figure 53: Snapshot of Training Dataset for PMV Index Prediction .....</b>	<b>86</b>
<b>Figure 54: Snapshot of Source Code for PMV Index Prediction Process using Weka.....</b>	<b>87</b>
<b>Figure 55: PMV Index Prediction Execution Result.....</b>	<b>87</b>
<b>Figure 56: Fuzzy Controller Variable Definitions .....</b>	<b>88</b>
<b>Figure 57: PMV Index Prediction Execution Result.....</b>	<b>89</b>
<b>Figure 58: Fuzzy Controller Fuzzy Rule Block.....</b>	<b>90</b>
<b>Figure 59: Experiment Scenario of the Smart Indoor Space Prototype .....</b>	<b>91</b>
<b>Figure 60: Intel Edison based Finalized Smart Indoor Space Prototype for Experiment ..</b>	<b>92</b>
<b>Figure 61: BPM Execution Result in IoT Proxy .....</b>	<b>93</b>
<b>Figure 62: Sensing Response Information in IoT Proxy .....</b>	<b>94</b>
<b>Figure 63: Built Linear Regression Model for PMV Index Prediction in IoT Proxy .....</b>	<b>95</b>
<b>Figure 64: Launch Fuzzy Inference System in IoT Proxy.....</b>	<b>96</b>
<b>Figure 65: Fuzzy Inference System Output in IoT Proxy .....</b>	<b>96</b>
<b>Figure 66: Smart Indoor Space Prototype Evaluation Structure.....</b>	<b>98</b>
<b>Figure 67: Smart Indoor Space Prototype Round Trip Time.....</b>	<b>99</b>

# List of Tables

<b>Table 1: Predicted Mean Vote Sensation Scale .....</b>	<b>48</b>
<b>Table 2: Nomenclature, description and measure units of the variables Involved in the PMV equation.....</b>	<b>49</b>
<b>Table 3: Considered fuzzy sets for input and output variables .....</b>	<b>56</b>
<b>Table 4: Sample of the fuzzy rules used .....</b>	<b>58</b>
<b>Table 5: Device Implementation Summary for Smart Indoor Space.....</b>	<b>63</b>
<b>Table 6: Logic Objects Implementation Summary.....</b>	<b>68</b>
<b>Table 7: Development Environment for IoT Smart Space Prototype.....</b>	<b>76</b>
<b>Table 8: Smart indoor space prototype performance.....</b>	<b>97</b>

# Abbreviations

API	Application Programming Interface
ARFF	Attribute Relation File Format
BPL	Business Process Layer
BPM	Business Process Modeling
BPMN	Business Process Modeling Notations
CoAP	Constrained Application Protocol
DIY	Do-It-Yourself
FIS	Fuzzy Inference System
HTTP	Hyper-Text Transfer Protocol
HVAC	Heating, Ventilation and Air Conditioning
IoT	Internet of Things
JSON	JavaScript Object Notation
LO	Logic Object
MQTT	MQ Telemetry Transport
POJO	Plain Old Java Object
REST	Representational State Transfer
VONL	Virtual Object Network Layer
SLL	Service Logic Layer

SCM	Service Composition Manager
SO	Service Object
SOA	Service Oriented Architecture
URI	Uniform Resource Identifier
VO	Virtual Object
VONL	Virtual Object Network Layer
VOM	Virtual Object Manager
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

# Abstract

In generally, Internet of Things refers to the networked interconnection of everyday objects, which are often equipped with ubiquitous intelligence. IoT will increase the ubiquity of the Internet by integrating every object for interaction via embedded systems, which leads to a highly distributed network of devices communicating with human beings as well as other devices. In recent years, IoT has gained much attention from researchers and practitioners from around the world. The Internet of Things (IoT) application development is a complex task that requires a wide range of expertise. Currently, the IoT development toolkit lacks support for inexperienced developers to develop IoT prototypes rapidly. Also, these systems do not consider the intelligent logic and business process. Filling this gap, a novel IoT cooperation architecture based on Business Process Modeling paradigm has been presented in this paper. The proposed IoT cooperation architecture comprises of physical cooperation network layer based on IoTivity and CoAP protocol, virtual object layer for virtualization of physical IoT devices, service logic layer for connectivity and intelligence of things, business process layer for developing process of IoT application and application layer for IoT application conceptual design. We also propose the logic object contains machine learning algorithm based elements for smart application in service logical layer. This study utilizes the existing concepts of virtual objects and service-orientation to enable end-users to configure and wire IoT service objects together to create an IoT application via simple actions like drag-n-drop and clicks etc. Through visual components, proposed system encapsulates the complexity of communicating with devices and services on the internet and abstracts them as virtual objects that are accessible through different communication technologies. In order to prove the feasibility of the proposed system, a smart indoor space prototype has been implemented for this purpose. First, we create virtual objects of physical temperature node, humidity node, wind speed node, fan node, and proxy node. Then we combine the virtual objects to generate service objects. We also implement PMV index calculation, linear regression prediction and fuzzy logic

based control module through which the users are able to combine with the virtual objects to configure complex service process for indoor environment automatic control. It allows participants to use the proposed architecture for service composition and BPM based process development for the smart space prototype. Well-established open IoT technologies such as OCF, IoTivity and CoAP protocol have been utilized in the proposed prototype application. The prototype has been experimentally tested in an indoor working place and the experiment results illustrate the feasibility and performance of the proposed IoT architecture.

# 1. Introduction

The Internet of things (IoT) is becoming an increasingly growing topic of conversation both in the workplace and outside of it [1]. IoT promises to enable building novel applications in areas such as building and home automation, smart environment, agriculture, intelligent transportation, and healthcare [2]. IoT is the basic concept of basically enabling a global connectivity of any devices with an on and off switch to the Internet between the real world and a virtual world. Such moving devices sense their environment and interact with their surroundings to take/give actuation orders. An increasing number of applications rely on such devices to offer people digital aids for their everyday activities. In this continuously evolving IoT landscape, recent studies show that the number of connected smart objects will radically increase in the next years [3].

With this blooming of smart devices, the challenge arises, how to put devices together to make IoT applications? This challenging issue manifests that many IoT infrastructures are built with a lack of interoperability in system integration such as data transmission, device management, and service composition and application deployment [4]. With the rapid increment in the dynamic IoT market, it is inefficient and time-consuming for developers to meet all these requirements as new products and techniques are arising all the time [5]. The number of communication protocols and IoT technologies are steadily increasing, and more and more APIs and data models are released for a specific domain and product. IoT mashup is a new solution for easing the development of IoT applications using familiar web development tools and technologies. Recently, the Representational State Transfer (REST) architecture has appeared, leading the development of Web of Things [7]. Things are identified by URIs and use a common protocol (HTTP) for stateless interaction between clients and servers. Using web protocols makes the creation of mashups possible allowing developers to combine data from both physical data sources and virtual sources on the web [8-9]. Various high-end IoT or M2M platforms and toolkits have been developed,

platform such as GeoThings [10] and Fitbit [11] providing Application Programming Interfaces (APIs) based on Representation State Transfer (REST) for easy device and app integration. openHAB [12] is an open source platform designed for smart home environment which provides Integrated Development Environments (IDEs), proprietary APIs, Software Development Kits (SDKs) and script engines. However, with the development of web front-end technologies such as JavaScripts and HTML5, these IoT platforms moving in a Web-centric direction with REST APIs and Web-based dashboards helping users to quickly set up and monitor the platform. In contrast, WoTKit [13] allows users to quickly find and subscribe to sensor data of interest, process data and visualize the data using widgets on a dashboard. Platforms such as Kaa [14], Predix [15] or Carriots [16] providing dashboards for data management of connected objects, SDKs for integration between device and server, web-based APIs for integration with product-specific services. However, the point is that all these solutions are perfectly valid for their specific application domains but limit to user customized domains. These platforms provide their own programming environment therefore users must be able to know programming languages such as Java or C to construct their applications. In this paper, we propose an enhanced IoT mashup platform for users who have no programming experience to design and develop their own IoT prototypes.

Our vision is to let the user model the process based on the available atomic services which have some level of composition and then integrate them with user defined rules to model their processes and directly execute those processes. Sometimes, these sample services can't satisfy the users' requirement such as in smart home environment since the cooperation between devices are required. To satisfy this requirement, machine learning algorithm approaches are implemented to integrate the logic process between multiple devices. Such an approach can enable the users to easily model and execute their desired processes and hence make IoT applications agile with respect to the user requirements. This work proposes the use of standardized Business Process



Modeling Notations (BPMN) based enhanced IoT architecture [17]. The proposed architecture is aimed at reducing the development difficulty and enable ordinary users to intuitively utilize virtual representations of IoT devices to compose services and to visualize those services as business process modeling notations to graphically model their own IoT applications or customize the existing ones according to their needs and requirements. This concept proves even more important in the case of IoT application development because the number of IoT enabling technologies is very high and it is still growing with time. In such a scenario if mass involvement of general population with lesser or no technical/programming skills is required, the DIY concept must be implemented in its entirety [18].

Figure 1 presents a generic comparison between the existing architectures which claims to provide a mashup approach for easing the development of IoT applications and the proposed IoT Architecture based on enhanced Business Process Modeling Approach. The first two layers of both the architectures have the same goals of representing the physical devices as virtual objects. In different systems, these virtual objects may be represented using different encoding technologies but nevertheless, the main objective of the virtual objects remains the same. Hence, the general functionality of the first two layers in the existing and the proposed architecture is same.

The third layer represents the Service Layer where the virtual objects are utilized to create service objects. The main aim of this layer in the existing and the proposed architecture is similar and that is to provide an intuitive approach to enable the users to create the services of their own choice and according to their own requirements. Small services are combined together to a large service, this process is called service composition. Simple logic is implemented through point-to-point exchanges or primitive compositions in service-oriented solutions. But the proposed system provides logic objects to define more complex variations between objects. Machine Learning algorithms based approaches are encapsulated as logic objects that define adaptive operations

between multiple objects. The existing systems mostly expose these user-created services as Application Programming Interfaces (APIs) and let the users create their own client applications by utilizing these exposed APIs. This allows for the creation of more requirement specific applications but it violates the basic concept of DIY development paradigm. The client application development with application logic at the client side is again a developer's job and people with no programming skills may not be able to create their own custom applications. At the end such application development is time consuming and the resulting applications cannot be modified easily.

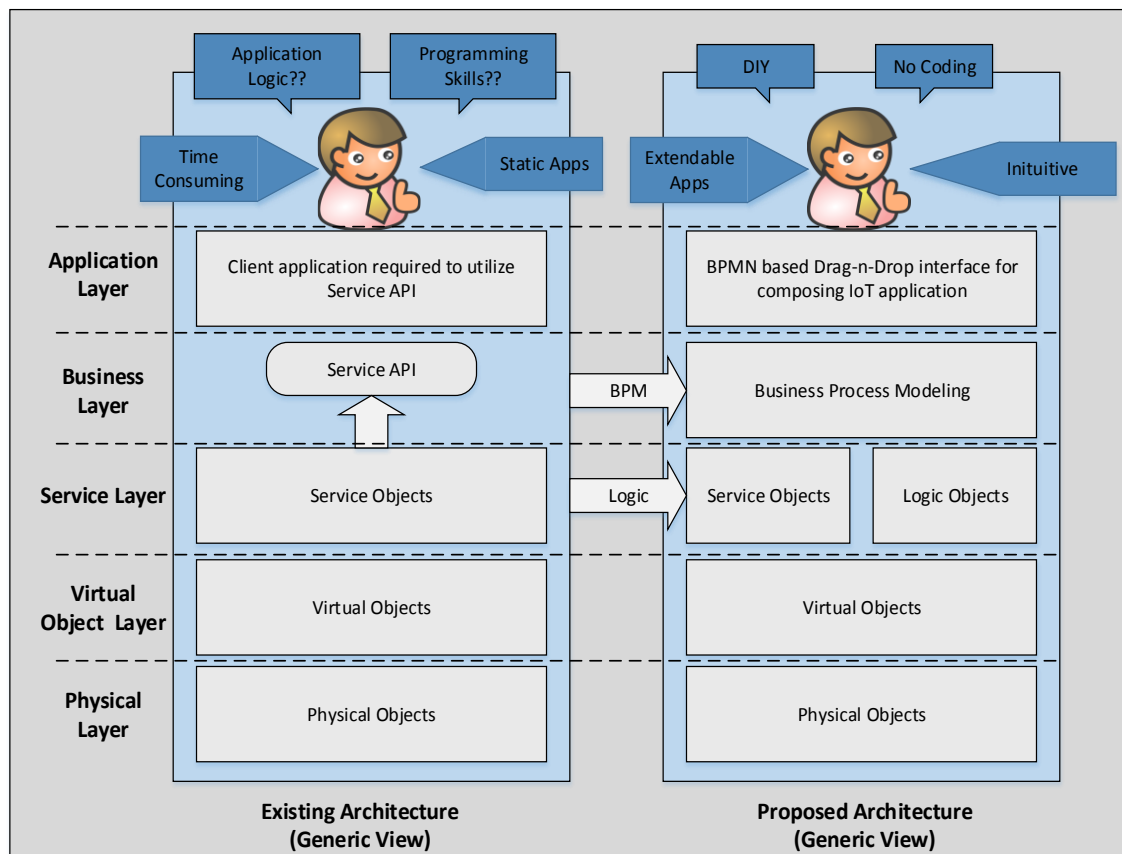


Figure 1: Comparison of the existing and proposed IoT cooperation system architecture

The proposed IoT composition architecture enhances the existing architecture by providing a Business Process Modeling approach for an IoT application development platform and supporting logic objects for defining complex composed service objects. The service objects created at the Service Layer are represented as standardized Business Process Modeling Notation (BPMN) along with notations to help create the application logic in an intuitive visual manner. The simple action such as Drag-n-Drop, mouse clicks etc. are well known to general population of modern age, hence enabling anyone to create their own IoT applications. No coding is required on behalf of the application developer (end-user) and the created applications can be easily modified through the manipulation of the business process model representing the application.

The next chapter of the dissertation provides a thorough analysis of the related projects and the existing IoT techniques (software, hardware, protocol etc.). Chapter 3 provides the details of the proposed architecture by explaining the major component and functionalities at each layer. Chapter 4 provides some insight into the implementation of the proposed system. In order to evaluate the utilization of the proposed architecture in different IoT scenarios, one use-case have been developed from the fields of smart spaces. Prototype implementations have been developed in order to evaluate the performance of the proposed architecture in this use case from the perspective of its usability.

## 2. Related Work

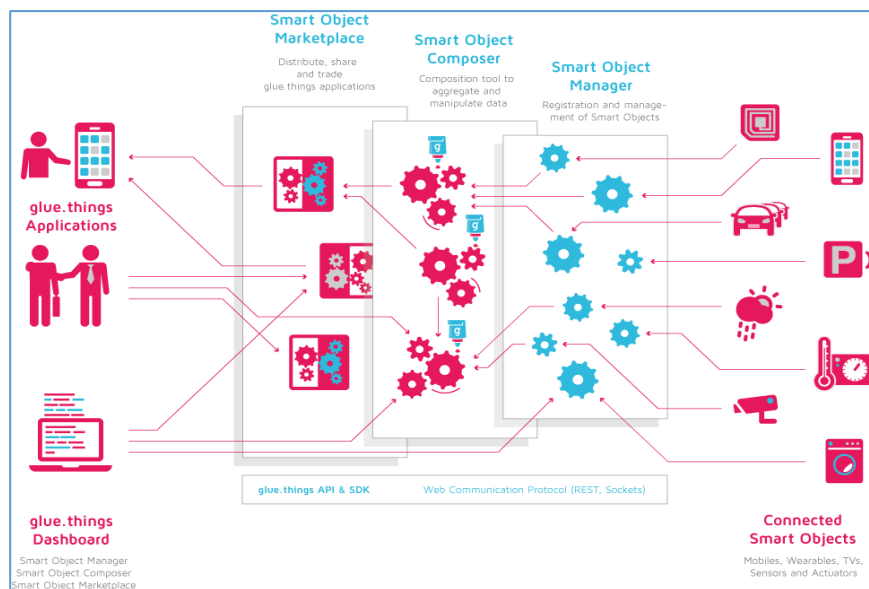
The next wave in the era of computing will be outside the realm of the traditional desktop. In the Internet of Things (IoT) paradigm, many of the objects that surround us will be on the network in one form or another. The IoT vision is the realization of a worldwide network of connected and interoperable smart devices which can provide services to the people. Although the individual technologies in terms of communication and devices have improved tremendously, the implementation and realization of such a complex network is still in its nascent stage. The end goal is to have plug-n-play smart objects that can be deployed in any environment with an interoperable 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 includes the coping with the heterogeneity of the hardware and mass involvement of general public in the IoT development and adaptation process. The first issue is being abstracted out with the help of 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 masses in the development of IoT.

### 2.1. Existing IoT Composition Platform

Here we discuss some of the recent IoT related projects which, in some way, provide a unique and alternate interface for end users to get involved in the design and development of IoT in our daily life.

Glue.things [19], is a recent project which 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 streams networks to allow mashups of the data streams,

add actions etc. The final mashups are deployable in a distributed environment. The system specially focuses on the composition of data streams from Web services and IoT devices with Web interfaces. The main aim of Glue.things as shown by the architecture in Figure is to utilize web technologies for providing interoperability platform with REST APIs, JSON data models and Web sockets etc. The Mashup interface is based on Node-RED [20] which is a browser based visual data stream aggregation tool. Another important aspect of the project is the utilization of well-established Open Source technologies.



**Figure 2: Generic architecture of the Glue.Things project**

IoT Mashup Application Platform (MAP) [21] is an effort towards flexible interoperability of smart things with smart phones in users' personal pervasive environments. IoT Map decouples the development of mobile application from the static model chosen by the designer/developer of the application which is a great hurdle in the way of application adaptability to the user's needs and/or the surrounding environment. IoT MAP utilizes the concept of abstracted service objects for the development of IoT applications. This is achieved through a set of application programming interfaces (API) exposed for functions like discovery and retrieval of the service objects etc. The business logic is written in POJO [22] while abstracting out the details of

connectivity and implementation of smart things. As shown in the conceptual architecture presented in Figure , the users can also utilize a graphical authoring tool based on NodeRed to compose an IoT application which can be converted into API calls and executed as an application.

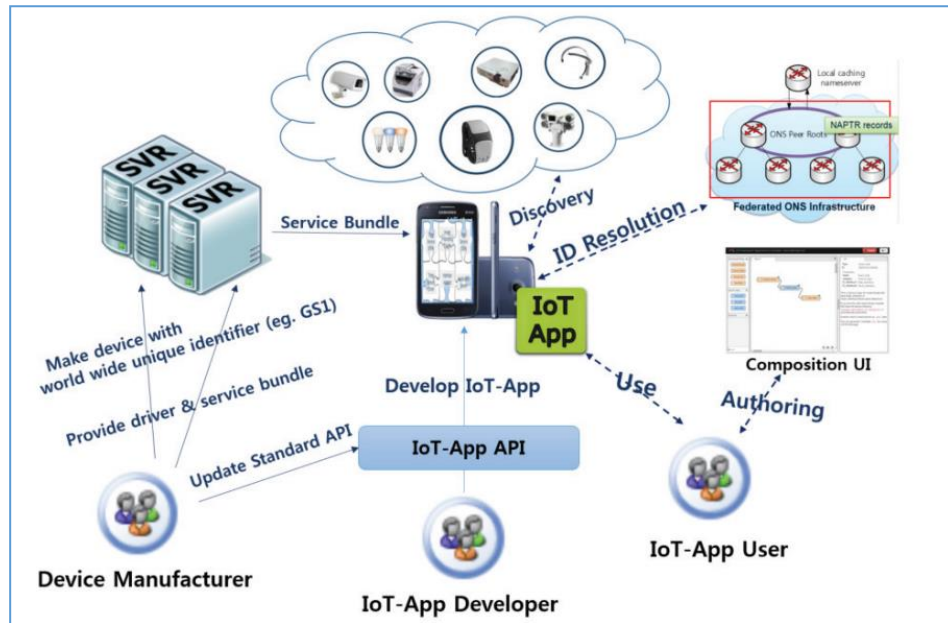


Figure 3: Conceptual architecture of IoT MAP

From the perspective of reducing the complexity of IoT application development, IoTLink [23] has been presented as a development toolkit. The IoTLink toolkit is based on a model driven approach, which utilizes a domain-specific graphical programming language to allow inexperienced developers to compose their own IoT applications. It encapsulates the underlying complexities of interaction with IoT devices and services into visual components. These visual components represent the IoT devices as virtual objects which can be accessed through multiple communication technologies. The high level architecture of the IoTLink project is shown in Figure . The project utilized IBM post-study system usability questionnaire to get user feedback regarding the system usability.

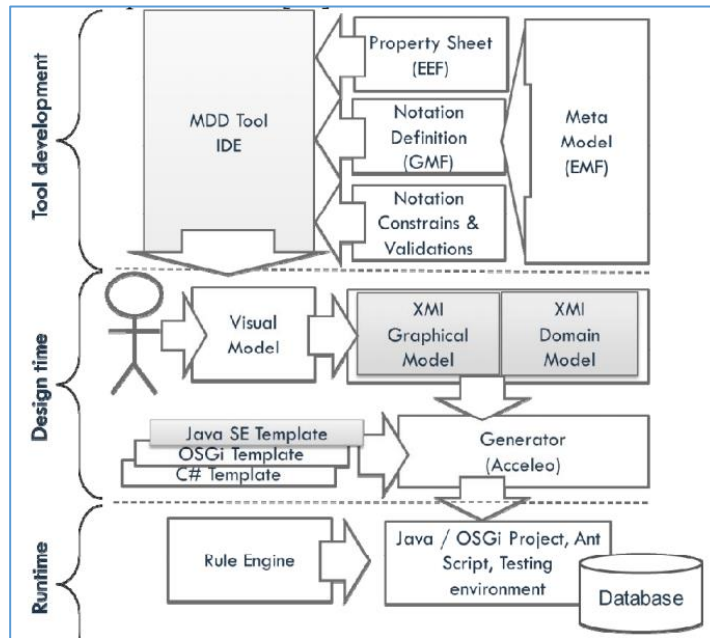


Figure 4: High-level architecture of the IoTLink

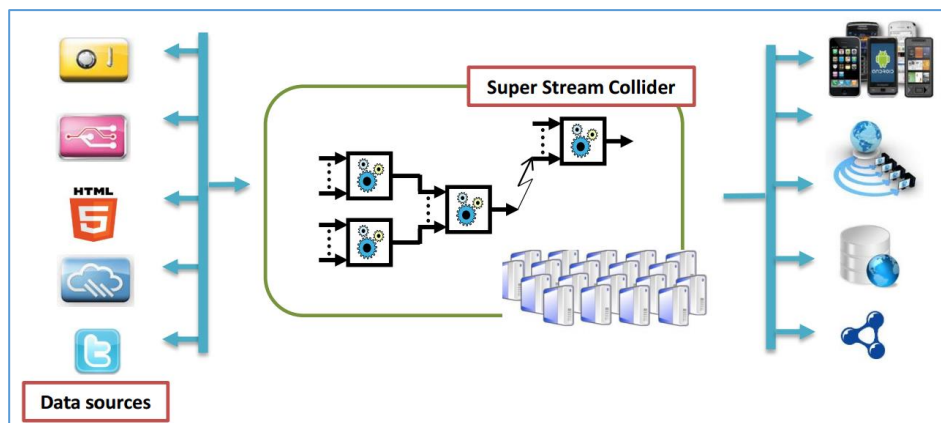


Figure 5: Layered architecture of the SSC platform

Super Stream Collider (SSC) [24] as shown in Figure is another platform which 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 which can be used as applications for IoT scenarios. The system supports drag-n-drop technique with a

SPARQL/CQELS 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. [26]. The visual development tools are intended to be used as an integrated development environment (IDE) for the support of IoT application development lifecycle. The tools presented are based on a semantic IoT architecture and claims to be a minimal programming environment for IoT application development. It uses a node-based user interface theme to allow the user to model service graphs and then convert them into SPARQL queries.

A multi-layer architecture framework has been proposed in [27]. The IoT architectural framework with configurable nodes and multiple sensors could be utilized in diverse applications. The sensor nodes collect data from the surrounding environment and pass it to the cloud for universal accessibility by the users. The proposed framework can be used for diverse applications and has been verified through hardware implementation for applications such as healthcare, wearables, structural health monitoring, object tracking, and connected vehicles.

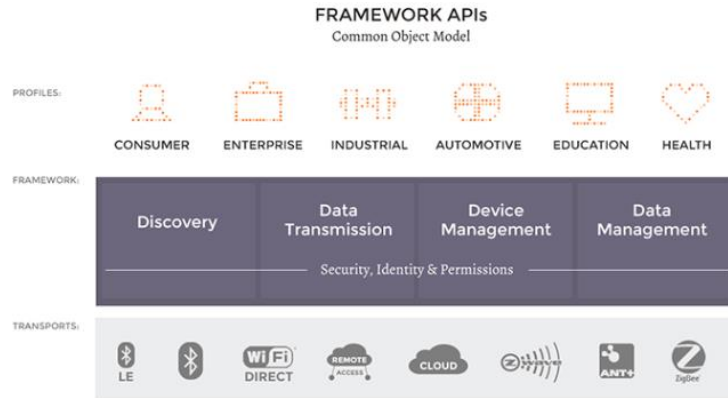
The OPEL software platform [28] is designed considering the following requirements. First, the IoT platform should be programmable with an easy programming language. In particular, the programming language should have high productivity, portability, and extendibility to enlarge the IoT ecosystem. A good candidate is 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, user can exploit multiple services on 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.



## 2.2. Existing IoT Open Source Framework

This section provides an overview of the common IoT open source frameworks. Here we discuss some of the widely used and popular frameworks for the Internet of Things.

IoTivity [29] is an open source software framework enabling seamless device-to-device connectivity to address the emerging needs of the Internet of Things. The IoTivity is sponsored by the OCF (Open Connectivity Foundation) who is developing a standard specification and certification program to enable the Internet of Things. This open specification is determined to unlock the massive opportunity in the IoT market, accelerate industry innovation and help developers and companies create solutions. The goal of IoTivity is to develop an open source software framework that can seamlessly connect billions of devices to the future of the Internet world, regardless of operating system and network protocols. One of the most important parts for activating the Internet ecosystem is how can small and medium-sized companies manufacturing various things add an Internet connection function to their products and provide an environment that can easily provide them with smartphone apps or services. IoTivity is a framework for satisfying these requirements, ensuring interoperability between high-quality Internet devices and developing high-speed internet products. It is also expanding the range of open source hardware (e.g. Raspberry Pie, Edison) and software platforms (e.g. Android, iOS, Windows, Linux, etc.) Currently, IoTivity supports Ubuntu, Tizen and Android, and iOS will be supported in the future. The open source hardware platform now supports Arduino and Edison, and will continue to expand its supported hardware platforms. Basically, IoTivity is an open source technology for Internet middleware based on OIC standards.



**Figure 6. OCF IoTivity architecture**

Figure 6 shows the conceptual architecture of IoTivity that consists of three layers. Transports layer supports the existing protocols such as Bluetooth, Wi-Fi, Zigbee, etc. Profile layer stands for each vertical field of object Internet applications such as smart home, smart factory, eHealth, etc. A framework layer supports functions such as resource discovery, data transfer, device management, and data management. In the case of the transport layer, new technologies can be continuously extended, and even with these extensions, the application layer of the profile layer can be executed without modification, with the support of the framework layer. For reference, the license policy follows Apache 2.0 and is operated by the Linux Foundation. IoTivity is based on a resource-based RESTful architecture model, thus representing all things as resources and providing CRUDN (Create, Read, Update, Delete and Notify) operations. In addition, it is designed based on CoAP (Constrained Application Protocol) without a daemon, so it is easy to support low-end and low-power devices.

A standard of M2M [30] is published by the oneM2M Global initiative which includes functional architecture, security solutions, protocol binding, ontology, etc. The documents that the initiative have published also involves solutions of interworking with other frameworks or platforms. oneM2M was initiated by seven telecom standards defining organizations: the Telecommunications Technology Association (TTA) in Korea, Association of Radio Industries

and Businesses (ARIB) and Telecommunication Technology Committee (TTC) in Japan, the Alliance for Telecommunications Industry Solutions (ATIS) and Telecommunications Industry Association (TIA) in United States, the China Communications Standards Association (CCSA) in China and the European Telecommunications Standards Institute (ETSI) in Europe [31]. Currently there are 238 participating partners and members, actively contribute to oneM2M. In addition, various alliances and industry fora such as the Open Mobile Alliance (OMA), Broadband Forum (BBF), Continua Health Alliance, and the Home Gateway Initiative (HGI) have recently become oneM2M partners. oneM2M defines three reference points [32] (i.e., Mca, Mcc, and Mcc') for communication between oneM2M entities as indicated in Figure 7. For example, the Mca reference point enables the communication between AEs and CSEs; Mcc is responsible for communication between CSEs in the same M2M service provider domain; Mcc' provides similar functions to Mcc but the communication is taken over different M2M service provider domains. In oneM2M, Mcc' plays an important role to provide an extension of service reachability from one service provider domain to multiple service provider domains. The Mca reference point needs to follow the oneM2M interoperability between AE and CSE to enable interworking between applications and a device/gateway/server. In the case of interworking between device and gateway or between gateway and server, the Mcc reference point should be oneM2M interoperable. The Mcc' processes communication between IN-CSEs in both M2M service provider domains for service provider communication in the same domain. Integrating multi-vendor solutions in the domain of IoT is common, essential, and complicated. This standards based interworking architecture can serve to reduce interworking complexity.

The open Home Automation Bus (openHUB) [33] is an open source framework aims to help implementing smart home application. openHUB is a software providing a single solution which allows over-arching automation and offers uniform user interfaces for integrating different home automation systems and technologies. It is a pure Java-based framework which can run on any

device capable of running a JVM. The modular stack abstracts all IoT technologies and components into “items,” and offers rules, scripts, and support for persistence -- the ability to store device states over time. OpenHAB offers a variety of web-based UIs, and is supported by major Linux hacker boards. The openHUB also provides user interfaces for mobile devices running Android or iOS system.

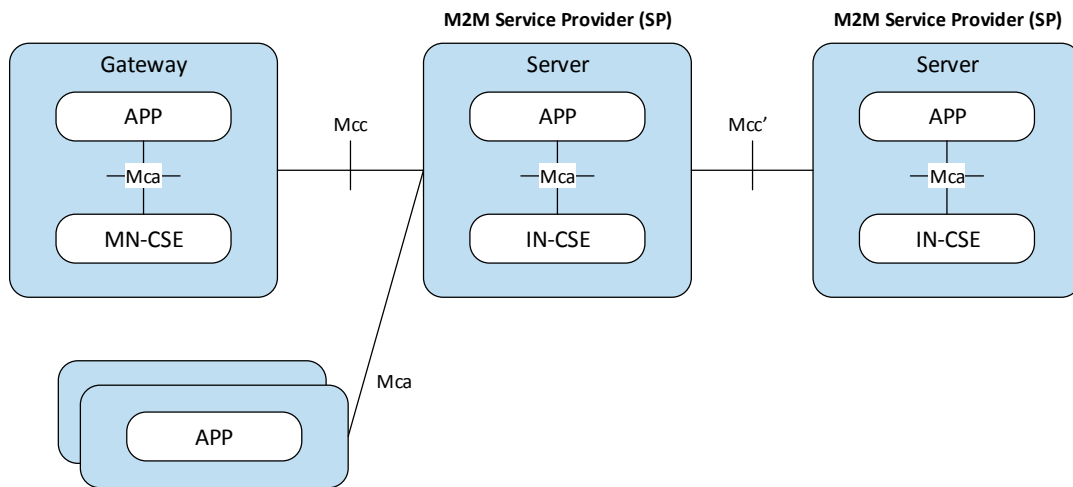


Figure 7. oneM2M reference architecture

The OpenIoT [34] is another Java-based open source platform for IoT, which aims to facilitate open, large-scale IoT applications using a utility cloud computing delivery model. The platform includes sensor and sensor network middleware, as well as ontologies, semantic models, and annotations for representing IoT objects. The OpenIoT is a unique open source platform for developing and integrating IoT applications. It supports the integration and use of both physical sensors and virtual sensors. OpenIoT blends IoT with semantic technologies by providing the means to use semantically rich descriptions about sensor data and metadata which complies the W3C SSN ontology. Furthermore, OpenIoT supports integration of data streams within both IoT

and cloud infrastructures [35]. It is built on open source technologies such as GSN (Global Sensor Networks), through which users can gain access to these technologies. OpenIoT extends the cloud computing implementations, which provides the approaches to formulate and manage increasingly important IoT based resources and capabilities which can deliver on-demand utility IoT services such as sensing as a service.

## 2.3. Existing IoT Open Source Hardware

Open source hardware consists of physical artifacts of technology designed and offered by the open-source culture movement. Open sources means the information about the hardware is easily discerned, so that anyone can learn, modify, distribute and manufacture. Open source hardware is usually made with components and materials that are easily available to each individual, standard processing methods, open facilities, unrestricted content, and open source design tools. This section introduces some of the recent and widely used IoT open source hardwares.

Arduino [36] is a tool developed by interaction designers Massimo Banzid and Davide Cuatielles for the purpose of pursuing an interactive design experience in an inexpensive way. As an open source computing platform based on a simple microcontroller board, it provides a development environment for interactive object software that receives inputs from various sensors and controls various outputs including motors, lights, etc. Features of Arduino include low cost, easy accessibility and infinite scalability. It can be purchased for as low as \$ 35 compared to other microcontroller platforms. While most microcontroller systems have limitations in the operating systems available, it can run on Microsoft Windows, Macintosh OSX, and Linux. It also provides a simple and easy programming environment, making it easy for non-technicians like designers to program. In terms of hardware, the module design is distributed under the Creative Commons License (CCL), allowing experienced circuit designers to create, extend, enhance and distribute their own module versions. Even if the circuit design is immature, it is possible to model the

prototype version with Arduino to reduce the operating method and cost. Arduino has a basic board type USB, Mega, Fio, LilyPad, Nano, Mini and a shield line which is based on this board and which is mounted on the existing Arduino board to expand its functions. Arduino boards are generally used for UNO and YUN, and TRE is in preparation for release. Figure 8 shows a representative UNO board as a business card size board. YUN is a Leonardo version that adds WiFi as a System on Chip (SoC) and has the same specifications as Leonardo.



**Figure 8. Arduino UNO**

Tre is a 1 GHz Sitara AM335x processor with 100 times the performance of Leonardo or UNO. For example, the development of the Arduino UNO requires the Arduino board and the "A plug to B plug standard USB (Universal Serial Bus) cable". The development tool provides the Arduino IDE (Integrated Development Environment) in [arduino.cc](http://arduino.cc). Since the power is supplied via USB cable, there is no need for separate power supply. PWR Power of board) It is possible to confirm power supply by lighting on LED (Light Emitting Diode). Install the ADOINO UNO driver, enable the COM port, and launch the ADOINO application. Software written using Arduino is called sketches and is stored with an .ino extension. In the Arduino application, select the board you want to use in the Tools> Board menu, in this case "Arduino UNO" and the COM serial port set above. Finally, the software is transferred to the board through the upload button of the application.

Raspberry Pi [37] is headquartered in the UK. Educational charity in computer and related disciplines focused on the introduction of low-cost computers for education that span both adults and children. Raspberry Pie is a credit card sized computer with a simplified circuit using the BCM2835 chip, which can be used as a PC by connecting a monitor and keyboard mouse. Learn how to program with scratch and Python, or use devices that are available to people studying computing, and do all the work on your existing desktop computer, including word processing, games, Internet browsing, and high-definition video playback. Above all, Raspberry Pi is able to interact with various fields of real life. With a variety of low-cost expansion tools, elementary and middle school students are actually making an interesting attempt in active fields. For example, it has been used in a wide range of digital maker projects, including music machines, parent detectors, bird house observations using weather stations and infrared cameras. Raspberry Pie offers two types of low-performance A-type and high-performance B-type boards and camera modules. Raspberry Pie has Radio Corporation of America (RCA) and High Definition Multimedia Interface (RCA) ports and audio ports for video output. It also has a local area network (LAN) port for communication and can be connected to a keyboard or mouse via a USB port. There are a number of GPIO (Genrel Purpose IO) and LEDs. The storage device has a SecureDigital (SD) card slot and 512 MB of RAM on the processor. The processor contains a CPU and a graphics processing unit (GPU). Raspberry pie development requires a raspberry chip board, SD card, monitor and connection cable, keyboard, mouse, and 5V Micro USB for power supply. If you are running Raspberry Pie for the first time, download the NOOBS (New Out Of Box Software) provided by Raspberry Pie to your formatted SD card and save it on your SD card. After preparation, connect the SD card and USB keyboard, mouse, monitor to raspberry pie in turn and connect Ethernet cable if you need to connect to internet. Finally, when you connect the power supply line Micro USB, the raspberry pie starts automatically. On the first boot, a list of installable operating systems starts. Raspberry pie recommends the Raspbian operating system, and you can

set the time, date range, etc. when you select it. The default account is pi / raspberry, and the process is the same as any other OS.

Edison [38] is the second open-source board from Intel following Galileo. It is targeted at companies of all sizes in the manufacture of home appliances or IoT and related industries, and low entry barrier to general purpose computing platforms. Edison is a small SD card size, low power, abundant functionality and ecosystem support inspires creativity and helps companies innovate with the rapid production of prototypes. Edison can be mounted in a standard SD card slot, and the SD card connector is compatible with SDIO (Secure Digital Input Output), so it acts as an existing SD card as an SDIO slave. As well as an SDIO host. The power supply also operates at very low power consumption with the voltage supplied to the SD card slot. Edison uses the next version of the Quark processor used in Galileo. Quark, powered by Galileo, was a single core in a 32nm process, but Edison uses a dual-core processor in an Intel Atom SoC based on the 22nm Silvermont microarchitecture. WiFi, Bluetooth Low Energy, memory and storage, and supports more than 30 industry standard I / O interfaces through a 70-pin connector. Software is also easy to develop with support for Arduino, Node.js, Python, Wolfram, and Yocto Linux.

## 2.4. Existing IoT Protocol

This section provides an overview of the recent and popular IoT protocols which are suitable for IoT application development. HTTP is the foundation of the client-server model used for the Web. The more secure method to implement HTTP is to include only a client in your IoT device, not a server. In other words, it is safer to build an IoT device that can only initiate connections, not receive. Although HTTP can be utilized as a reliable protocol for IoT implementations but due to its heavyweight protocol stack, it is not suitable for the resource constrained IoT devices.

XMPP (eXtensible Messaging and Presence Protocol) is an international standard protocol established by the Internet Engineering Task Force (IETF) for instant messengers. XMPP is



known as an open protocol for XML-based Internet communication. It is a protocol that is useful for exchanging current status information with messages that can communicate with other users in real time. It has expanded into signaling for VoIP, collaboration, lightweight middleware, content syndication, and generalized routing of XML data. It is a contender for mass scale management of consumer goods such as washers, dryers, refrigerators, and so on. XMPP is based on the open-type protocol Jabber Instant Messenger. Jabber XMPP uses a unique name for the client through the associated server communicates with other clients. Each client implements the client of the protocol and the server provides the routing function. XML stream must be a TCP socket. You can connect via HTTP polling or some other mechanism without needing to connect to it. The server can route XML data to the appropriate addresses between domains. The gateway also provides translation between external messaging domains and protocols.

MQ Telemetry Transport (MQTT) is an open source lightweight Publish / Subscribe protocol made for the purpose of using as a machine, machine-to-machine (M2M) and IoT. In addition, it is designed to be used in low-power and low-bandwidth environment implementing bidirectional pub/sub messaging service for efficient communication with mobile devices, and simplify the development between heterogeneous platforms. MQTT has the advantage of being able to operate on a low-power, unreliable network, and no TCP/IP support. It is advantageous for small device control and sensor information collection. These features are attracting particular attention in the IOT area. Facebook Messenger has already used it to service it, and Korean telecom companies are also using MQTT as a push service. M. Prihodko compared the processing between MQTT and HTTP to measure and analyze battery consumption and performance, and introduced the merits of MQTT. It helps minimize the resource requirements for your IoT device, while also attempting to ensure reliability and some degree of assurance of delivery with grades of service. MQTT targets large networks of small devices that need to be monitored or controlled from a back-end server on the Internet. It is not designed for device-to-device transfer and it is not

designed to “multicast” data to many receivers. MQTT is extremely simple, offering few control options.

CoAP is a lightweight version for limited Wireless Personal Area Network (WPAN) devices that cannot be used due to resource limitations. Consider payload maximum size of 127 bytes for 802.15.4. Reduce as much as possible to reduce header fields and so on, encode in binary, use simple UDP, encrypt in datagram unit using RFC 6347 DTLS, and use multicasting. It provides a discovery function, and it can be regarded as a publication-Subscribe event method in addition to the polling method of the request-response. It is possible to operate it in the public network, but it can be said that it is used only to the gateway connected to the WPAN, and the upper layer is used to utilize the existing web framework by converting into HTTP.

CoAP is an open standard and not proprietary like some of the earlier protocols for networked embedded systems [39]. The open standard means that the standardization process is open to public and that it is free to be used by anyone without any royalty. This fact alone makes it perfect for the global implementation of the Internet of Things.

The CoAP protocol has been designed with the focus towards resource constrained devices associated with IoT. Thus it is light in comparison to other IoT protocols. It is based on the same principles. Like HTTP thus it is very easy to use. It provides datagrams based asynchronous communication which is suitable from the perspective of constrained devices because it is very lightweight in terms of resource consumption.

CoAP runs over IP where IPv6 is the future of IoT. This feature enables the future IoT to easily integrate with the current IP based IT infrastructure of organizations and personal spaces. Using the IP based communication infrastructure, CoAP can be utilized for interconnecting the IoT devices with the HTTP and RESTful web and this can be done through simple proxies.

The CoAP is still in the standardization phase, new features are constantly being added to the protocol stack. The emerging CBOR encoding for CoAP has proved to be a better suit for REST than the conventional JSON and HTTP [40]. Since then, IETF has been using it as a standard protocol. The open standard means that the standardization process is open to public and that it is free to be used by anyone without any royalty. This fact alone makes it perfect for the global implementation of the Internet of Things. IoT standardization organizations such as oneM2M, Open Interconnect Consortium is also working on the standardization of IOT environment based on CoAP. OneM2M is a common service platform that supports IoT application services based on the CoAP protocol. In addition, OIC developed an open source project called IoTivity based on CoAP, and defined the resource type, search procedure, and security specification for each device. Recently CoAP interoperability test has been used as a communication protocol suitable for IoT environment by interoperability between devices.

## 2.5. Existing Open Source Machine Learning Software

TensorFlow [41] is an open source software library for machine learning used in Google products. The Google Brain team created TensorFlow for research and product development and was released on November 9, 2015 as the Apache 2.0 open source license. This is a second-generation machine learning system that follows Google's first-generation machine learning system, Disturbill, which has been in use since 2011. Because it is open source software, it can be used by anyone who wants, such as students, developers. Google announced that TensorFlow could run on smartphones and thousands of computers in the data center, a technology that can be used flexibly without any restrictions. It is open source software announced by Google, so it applied to Google search, voice recognition on Google app, 'smart reply service' which reads mail from G mail and provides example reply that is suitable for the situation. TensorFlow is a machine learning system that supports large-scale training and reasoning. TensorFlow uses the data flow graph to represent the calculation and state of the algorithm. It maps nodes of the data flow graph

to multiple machines in the cluster, as well as machines across multiple computing devices, including multi-core CPUs, general purpose GPUs, and custom design ASICs (called Tensor Processing Units (TPUs)). It can effectively use hundreds of powerful (GPU) servers for rapid training and run a trained model for production reasoning on a variety of platforms, from large distributed clusters in the data center to mobile devices running locally. Unlike traditional data flow systems where graph vertices represent functional calculations for immutable data, TensorFlow allows vertex representations to have or update variable state calculations. The edge carries a tensor (multidimensional array) between the nodes, and TensorFlow transparently inserts the appropriate communication between the distributed subcomputers. By unified computing and state management in a single programming model, TensorFlow allows programmers to try different parallelization scenarios, such as unloading calculations to a server that maintains a shared state to reduce network traffic. TensorFlow supports a variety of applications, focusing on deep neural network training and reasoning.

WEKA [42] is a software for machine learning and data mining developed in the Java language at the Department of Computer Science, University of Waikato, New Zealand. This tool is the most downloaded software in the ACM CHI Society and has received special awards. Data Mining by Professor Witten and Dr. Frank: practical machine learning tools and used in conjunction with techniques. The main functions of WEKA are both command line and GUI environment. It provides data preprocessing function, learning algorithm and evaluation method, learning algorithm comparison function, and visualization function that visualizes processing results. WEKA handles a simple flat file called arff. It is also possible to save a Microsoft Excel file in csv format and convert it back to arff. All training methods in Weka can be accessed from the command line, as part of the shell scripts, or from other Java programs that use the Weka API. Weka also contains an alternative graphical user interface, called the "knowledge stream", which you can use instead of a conductor. It is focused on a process focused on the process of data mining,

where individual learning components (represented by Java components) can be linked graphically to create a "flow" of information. Finally, there is a third graphical user interface - "Experimenter" - which is designed for experiments that compare performance (several) learning patterns with (multiple) data sets. Experiments can be distributed to several computers with remote experiment servers. The WEKA provides a Java interface so that WEKA's functionality can also be integrated into any Java application. Weka.jar includes all packages, which comes with the installation package available on the WEKA homepage. Implementations of the algorithms are also available so that they can be studied and modified.

Deeplearning4j [43] aims to be advanced plug and play, more convention than a configuration that allows you to quickly prototype for non-researchers. DL4J is configured by scale, all DL4J derivatives belong to their authors. DL4J can import neural network models from most of the basic structures via Keras, including TensorFlow, Caffe, Torch and Theano, overcoming the gap between the Python ecosystem and the JVM using cross-command tools for scientists, data engineers and DevOps. Keras is used as the Python API Deeplearning4j.

Lasagne [44] is a lightweight Python library built into Theano that simplifies the creation of neural network layers. Similarly, Keras [45] is a Python library that runs on Topo de Theano or TensorFlow (Chollet, 2015), which allows you to quickly define a network architecture in terms of layers and also includes features for image and text pre-processing. CuDNN is a highly optimized GPU library For NVIDIA units that allow for faster training of deep learning networks. It can greatly speed up the performance of a network in depth and is often referred to by other packages above.

### 3. Proposed IoT Cooperation Architecture and Composition System

The proposed IoT cooperation network consists of 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 represent as virtual objects which 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, these devices receive and parse the deployment processing information from virtual domain and then operate accordingly.

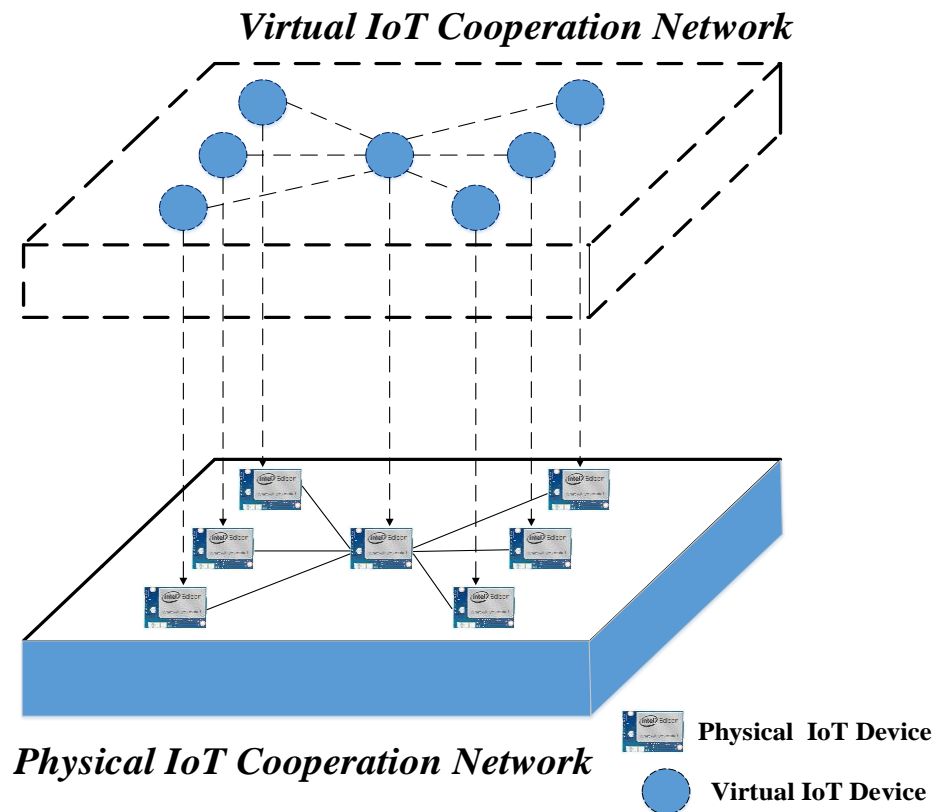


Figure 9: Conceptual physical/virtual IoT cooperation network

Figure 10 illustrates the role of the Integrated IoT Proxy in proposed IoT cooperation network. IoT devices in physical domain act like server and they require client to request the corresponding functionality (e.g., reading sensing data from sensors or controlling the actuators) of the resources associated with the server, the proposed architecture is based on the IoT proxy which acts like the operating client for the connected IoT devices. The proxy decouples the IoT devices from the virtual domain and it acts as the CoAP client and controls the operation of the connected devices as specified from the virtual domain. The IoT proxy communicates with an IoT device through CoAP / IoTivity protocol commands. For the connection to establish, the proxy must know the initial information about the IoT device.

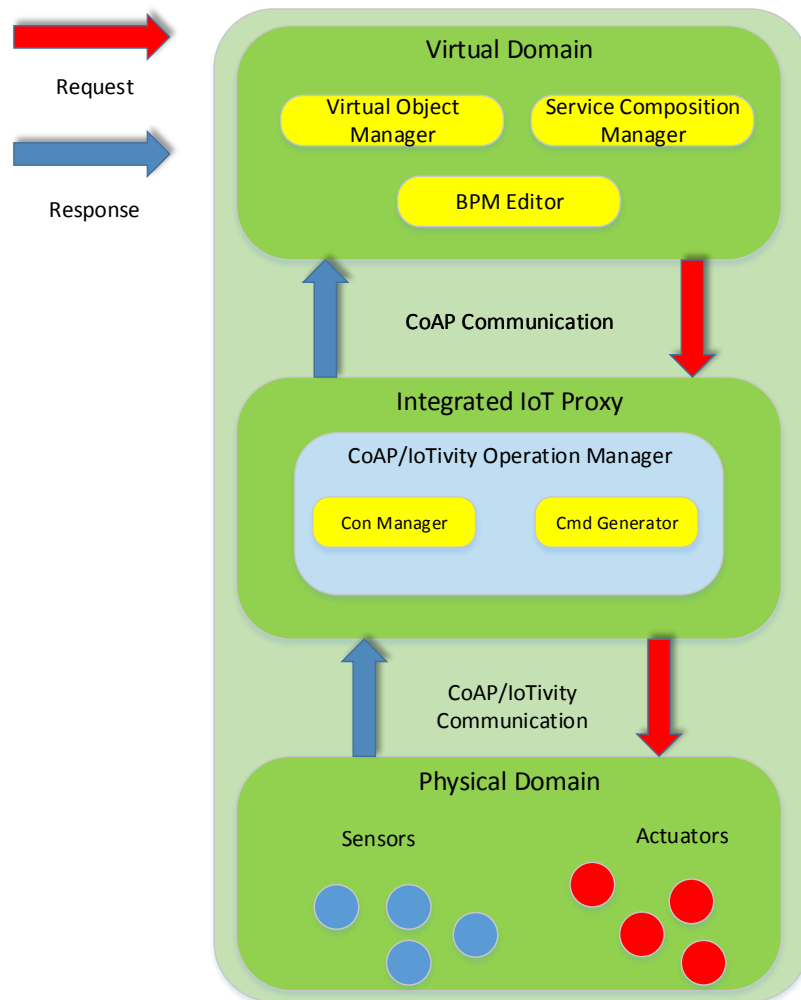


Figure 10: Connection between Virtual Domain and Physical Domain

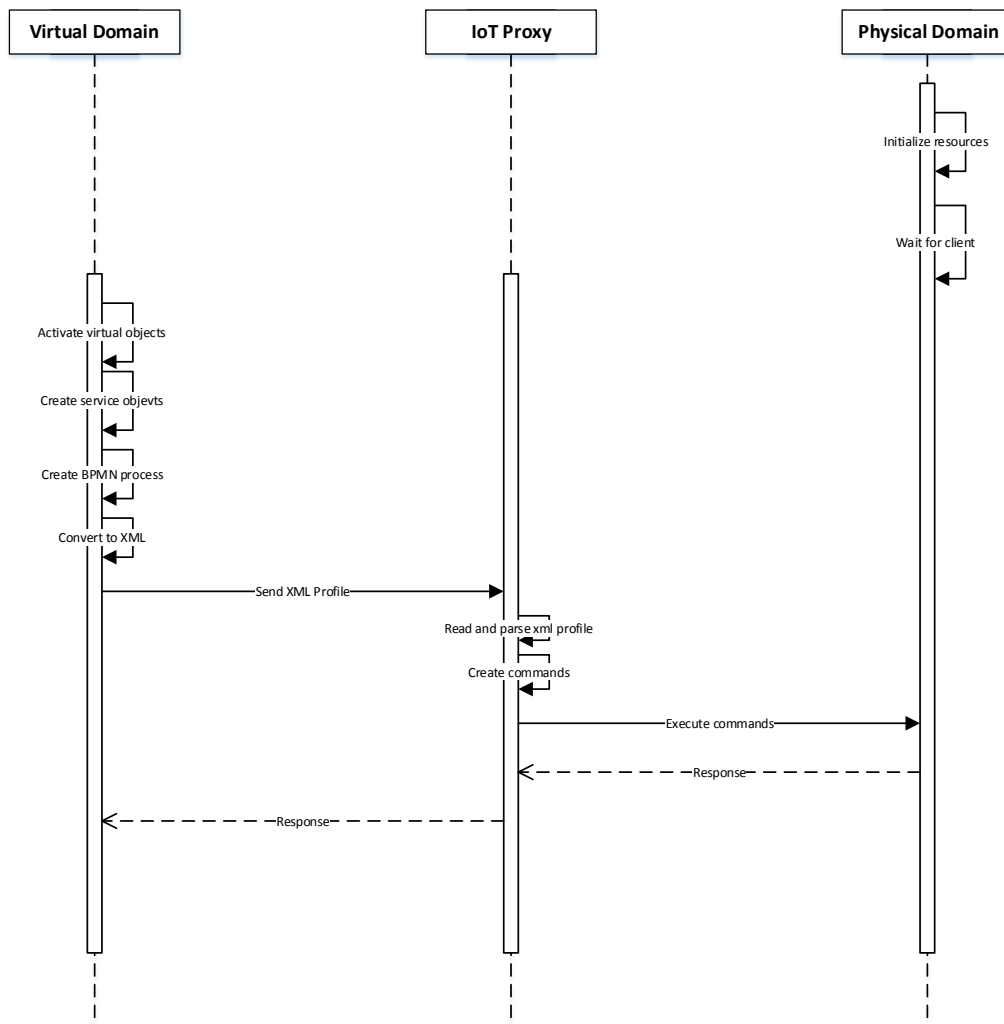
This initial information is the device IP address and CoAP port for the said server device. The proxy, using the IP and port, establishes a connection with the server device and retrieves the information about the available resources from the server in the form of a string. This information about the resources on the CoAP server is then communicated to the virtual domain which initiates the designer interface with the virtual representations of the available resources. The virtual representation of the IoT resources is termed as virtual objects (VOs) in the virtual domain. The communication between visual domain and the IoT proxy takes place using CoAP protocol. In this communication, the IoT proxy acts as a server while the virtual domain acts as client entity. The user r then uses these VOs to create a service flow which describes the functionality of the device server at virtual domain in terms of the available resources. The service flow is termed as the profile which is an XML representation of the graphical design. The IoT proxy uses the XML profile from the virtual domain to extract the role of each CoAP resource by parsing the XML profile and then dynamically translate the roles into CoAP/IoTivity commands. These commands are then executed on the remote device server.

Figure 11 illustrates the sequence of connectivity among the Physical Domain, the IoT, and the Virtual Domain. The figure also provides a general overview of the operation of the while configuration. The Physical Domain consists of the physical device which is intended to be operated via the IoT Proxy according to the graphical service design created through the Visual Domain. For this purpose, these physical devices must be running with a known IP address and port. The IoT Proxy connects as a client with the devices (CoAP /IoTivity server) using the IP and port number.

Virtual Domain provides a graphical interface for users to customize IoT applications. The users can represent the physical devices from the physical domain as virtual objects which encapsulate the behaviors of the physical devices. And then combine these virtual objects to generate service objects. The BPMN process is then created by the users to utilize these service



objects to customize the application process which is further deployed to control the Physical Domain. The BPMN process is converted into XML format and sent to the IoT Proxy. The design is saved as an XML file and can be reopened for updating the design flow. Once the visual design is complete, the user can send the XML profile to the proxy. Proxy reads and parses the XML profile to generate appropriate CoAP/IoTivity commands in order to operate the remote IoT device according to the service process created by the user.



**Figure 11: Sequence of the IoT Architecture based on IoT Proxy**

The following section is dedicated to the description of the proposed IoT system architecture. The details of each layer are divided into shared components and layer specific or application

specific components of the system. Each layer is described in terms of components and the activities performed by those components. The following text provides the description of each layer. Figure 12 shows the detailed architecture of the system.

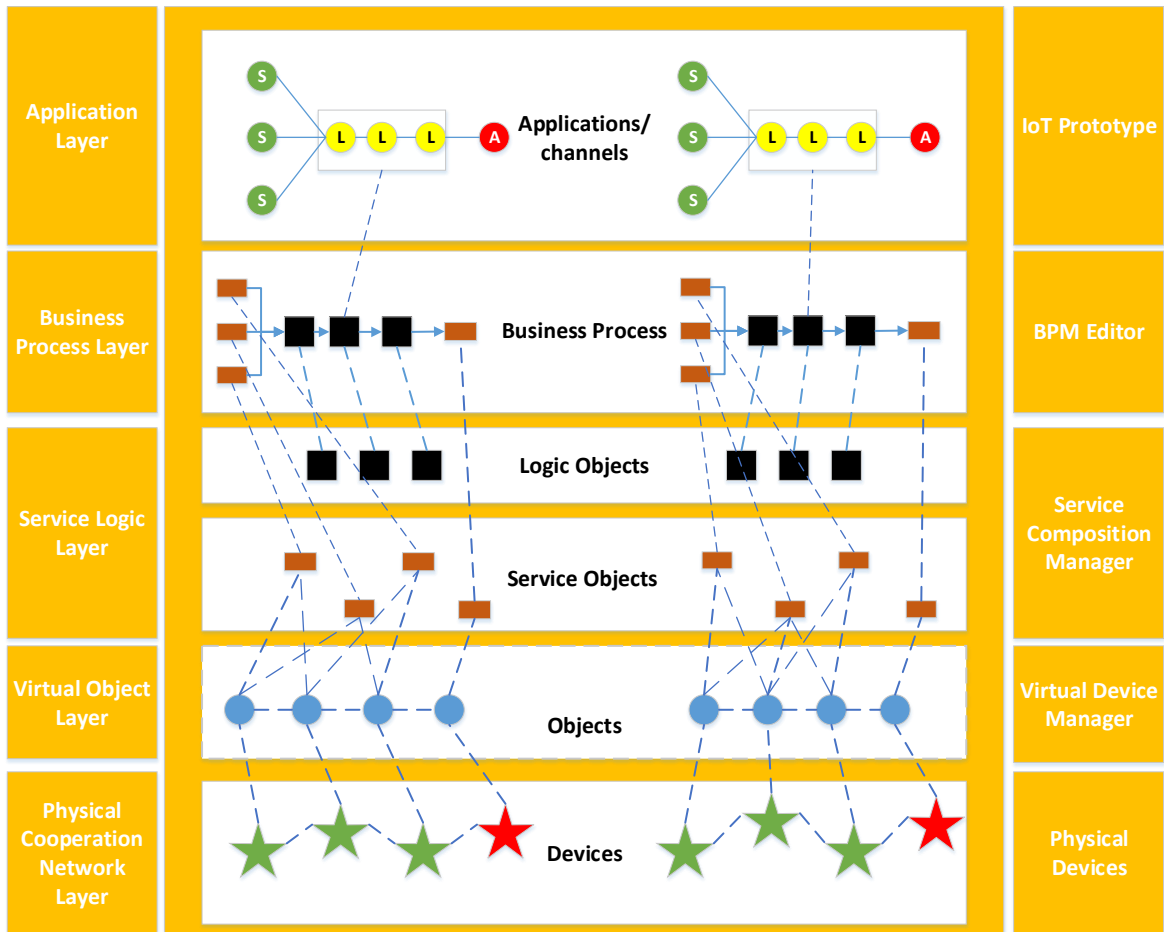
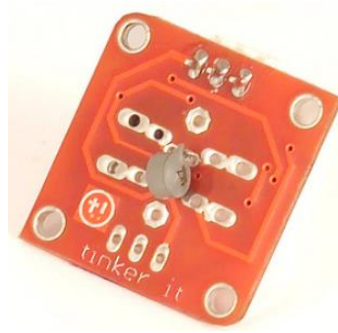


Figure 12: Proposed IoT Composition Architecture

### 3.1. Physical Cooperation Network Layer

The physical cooperation network layer consists of the following IoT sensors and actuators. The Thermistor is a resistor whose resistance varies significantly (more than in standard resistors) with temperature. This module's output approaches 5v as the temperature increases. As the temperature decreases, it approaches 0V. When connected to an input on the

Arduino using the TinkerKit Shield, expect to read values between 0 and 1023. This module features a Thermistor, a signal amplifier, the standard TinkerKit 3pin connector, a green LED that signals that the module is correctly powered and a yellow LED whose brightness changes according to the temperature.



**Figure 13: TinkerKit Thermistor Module**

The HIH-4030/4031 Series Humidity Sensors are designed specifically for high volume OEM (Original Equipment Manufacturer) users. Direct input to a controller or other device is made possible by this sensor's near linear voltage output. With a typical current draw of only 200  $\mu$ A, the HIH-4030/4031 Series is often ideally suited for low drain, battery operated systems. Tight sensor interchangeability reduces or eliminates OEM production calibration costs. Individual sensor calibration data is available. The HIH-4030/4031 Series delivers instrumentation-quality RH (Relative Humidity) sensing performance in a competitively priced, solderable SMD. The HIH-4030 is a covered integrated circuit humidity sensor. The HIH-4031 is a covered, condensation-resistant, integrated circuit humidity sensor that is factory-fitted with a hydrophobic filter allowing it to be used in condensing environments including industrial, medical and commercial applications. The RH sensor uses a laser trimmed, thermoset polymer capacitive sensing element with on-chip integrated signal conditioning. The sensing element's multilayer construction provides excellent resistance to most application hazards such as condensation, dust, dirt, oils and common environmental chemicals.



**Figure 14: Humidity Sensor HIH-4030**

The three type wind speed sensor is an instrument which can measure the wind speed. It is composed of shell, the wind cup and circuit module. Photovoltaic modules, industrial microcomputer processor, the current generator, electric current and so on are integrated in the internal drive. The materials of sensor shell and wind cup is the aluminium alloy which use the special mold precision casting technology, the size of the tolerance is very small, the precision of the surface is very high, and internal circuit has been protection processing, the sensor has high strength, weather resistance, corrosion resistance and waterproof. The plug of the cable is a military plug, it has a good anticorrosive and prevent erosion performance that it can ensure the instrument used for a long time, at the same time, In the case of using relevant specifications which ensure the accuracy of the wind speed acquisition.

The material of the circuit pcb is the military grade A which ensure the stability of the parameters and the quality of the electrical properties; Electronic components are all imported industrial chip which makes overall has extremely reliable electromagnetic interference resistance, and can ensure that the host can work normally in - 20 °C ~ + 50 °C, humidity 35% ~ 85% (condensation). This product can be widely used in engineering machinery (crane, crawler crane, door crane, tower crane, etc.), railways, ports, docks, power plants, meteorological, cableway, the environment, greenhouse, breeding, air conditioning, energy monitoring, agriculture, health, clean room areas such as wind speed measurement, and the corresponding signal output.



**Figure 15: Wind Speed Sensor (SKU: SEN0170)**

L9110 Fan Module for Arduino Robot Design and Development Features: L9110 drive, can control positive and negative going motion .Equipped with installing hole, compatible with steering gear tiller control .High quality propeller, high efficiency .Can be easily blow out the lighter flame (beyond 20 cm) .Can be used for fire the robots and necessary development for firefighting, robot design and development necessary.



**Figure 16: Arduino L9110 Fan Module**

Figure 17 represents the structure of the physical IoT cooperation network. Three IoT sensors that introduced previously have been integrated with the Intel Edison board respectively and connected with the IoT proxy which is also implemented within the Intel Edison. The IoT Proxy

gets the sensing data from these three sensors and then generates the command which is sent to the fan actuator to control the fan speed.

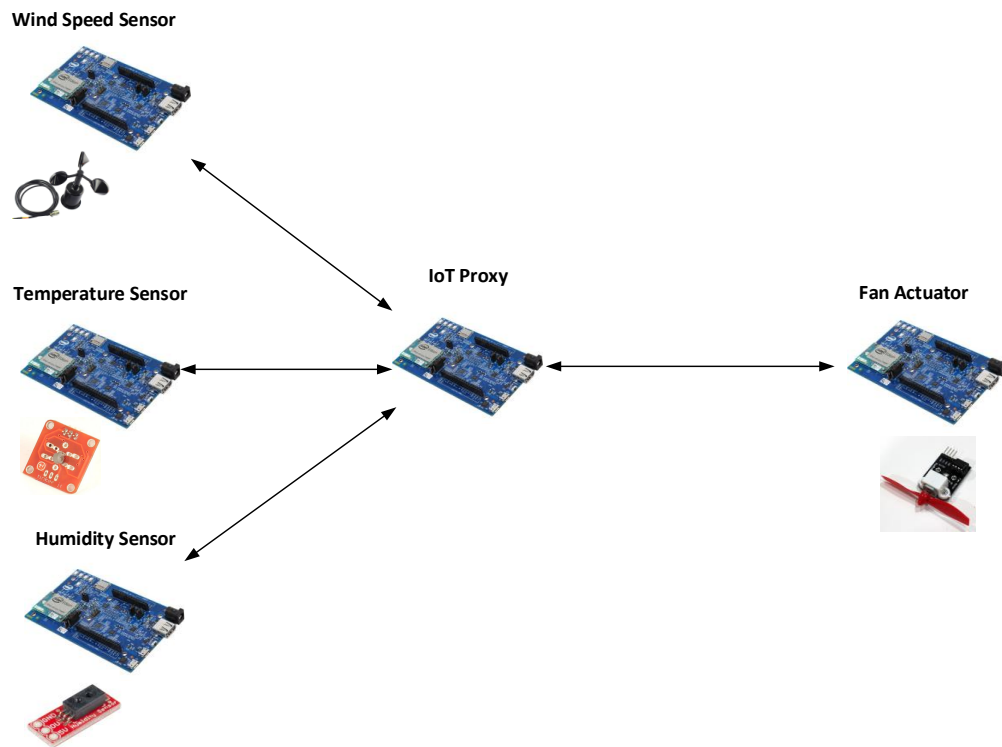


Figure 17: Physical IoT Cooperation Network Structure

### 3.2. Virtual Object Network Layer

The Virtual Object Network Layer (VONL) includes the Virtual Device Manager from [46] which represents the physical things in the form of VO Behavior, VO attributes and VO visualization. The VO Behavior is the services or functions which can be utilized by the system to interact with the physical thing represented by the VO. A simple example would be the name of CoAP service which can be called remotely to interact with the physical thing. The VO Attributes are the other information in the form of complete URI and Location etc. which collectively describes the existence of the physical thing through its virtual representation. Finally, VO Visualization is the graphical representation of VO depicting the type of the physical thing

represented through it. It is an icon or string of characters visually representing the underlying physical entity such as thermometer icon to represent a temperature sensor.

The VO information acquisition interface is used to register physical things as virtual objects. This interface can be a local data entry interface for a user or administrator to register the available physical things or it can be exposed as an online service to enable users to remotely access and register their devices. After the registration, the information related to VOs is stored at the VO Repository. VO Repository holds the information about virtual objects in XML document for so that it can be transferred easily over the Internet.

The Virtual Device Manager is the main component at the Virtual Object Network Layer (VONL). It in collaboration with other classes such as the File Manager, Communication Manager and Parser etc., provides the implementation of all the functionality associated with the VONL.

Figure 18 shows the startup and operational configuration model for the Virtual Device Manager. The local and remote interfaces are used by the users to enter information related to any CoAP enabled physical thing about which they have the required information. These interfaces can also be used to add new VOs, Delete and update the previously registered VOs. Once a VO is created for a physical thing, the visual representation, the attributes and the behavior of the physical thing are encapsulated in this virtual object.

The visual representations for the VOs already registered can be viewed as a list in the main interface from where the user can view the information associated with a VO, update and delete any information for the VO. The XML Parser is used to convert the VO into XML schema already defined for the VO representation. This XML version of VO is stored in the XML repository using the File Manager component. In order to send or transfer the VO information to a remote requesting component, the Communication Manager component through the File Manager reads all the VOs and sends it to the requesting component according to a predefined information exchange scheme.

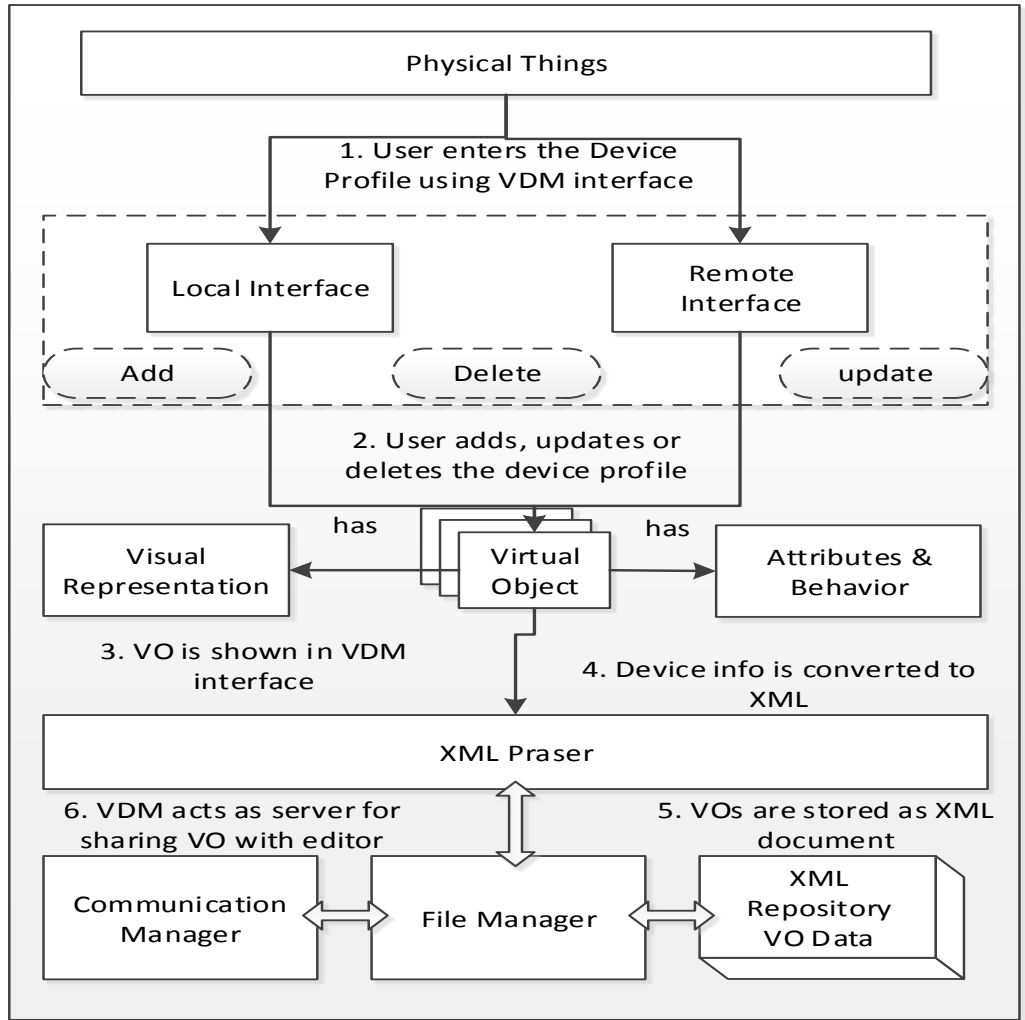


Figure 18: Virtual Device Manager Operation Configuration

### 3.3. Service Logic Layer

The Service Logic Layer (SLL) is the part of the system where the unit services are composed based on the information obtained from the virtual objects provided by the VONL. The Service Composition Manager (SCM) from [46] is responsible for providing an intuitive and easy to use visual environment where the VOs obtained from VOM are rendered as graphical module, enabling the users to drag-n-drop modules onto the main canvas and join them to create service objects. We updated this software by implementing logic objects in the service logic layer



repository which are represented as VOs. A Service Object consists of an Input VO joined to an Output VO through the Join element. Each SO is represented as a combination of Module elements and a join element in an XML document which is stored at the SO Repository.

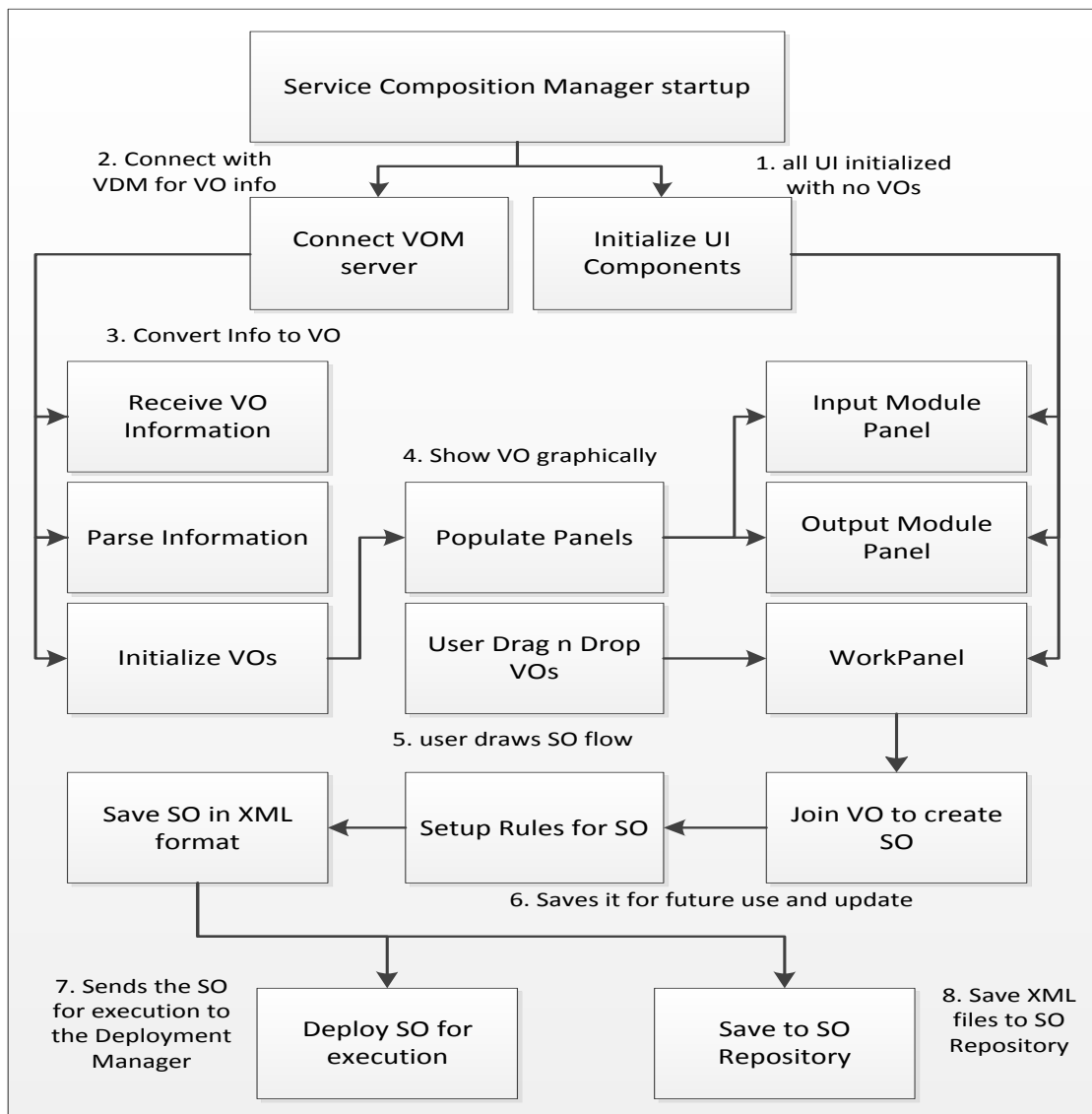
The Service Objects created at the Service Composition Layer are stored as an XML repository at the SCL. The SOs are designed in two types, one type is the independent SO which contains all the required entities to be runnable as an isolated process. The second type of SO are the one which define partial functionality (Unit SO) and can be combined into sequences to create larger processes.

Service Composition Manager (SCM) is the main component at the service logic layer. All the other components at the service composition layer are implemented as part of the SCM. The SCM is a DIY graphical designer that is used to compose service objects (SO) from the virtual objects (VO) and the associated information that is received from the Virtual Object Network Layer. The SCM startup configuration is provided in

**Figure 19.** At startup, the SCM initializes all the user interface components including the Input Module Panel, the Output Module Panel and the Work Panel. Based on the user's choice, a client component sends a connection request to the communication manager at the Virtual Object Layer. If the connection is granted, the SCM receives a list of virtual object information. The information includes resource name, URI, attributes and functionalities which can be remotely executed and information about the visual representation of the device. The information is sent as XML strings and upon reception by the SCM, it is parsed to initialize VOs. The virtual objects created are used to populate the input and output panels so that the user can visualize them and ultimately interact with them to compose them into service objects. Logic objects are defined in the repository of the Service Logic Layer which provide machine learning algorithm based services to define the logic

condition between VOs. Once all the available VOs have been instantiated, the user can interact with the SCM interface to compose Service Object (SO).

As SCM is designed to be a DIY service composer interface, the users perform simple drag-n-drop, click and double-click operations using a mouse pointer to compose a design on the WorkPanel, which acts as the main drawing canvas. The service composition process includes joining the input and output modules.



**Figure 19: Service Composition Manager Operation Configuration**

An example of this process can be visualized as the fan control service where the temperature sensor is the input VO, fuzzy controller is the LO and the fan is the output VO. The join between the VO and LO then specify the function such as reading from the temperature sensor. And then the fuzzy controller LO computes the output value that is the input value of the fan VO. The join between the fuzzy controller LO and the fan VO will then specify the function such as controlling the fan actuator. Once the service composition is completed, the visual SO is converted into XML data and stored at the SO repository. The SOs can also be deployed and executed for testing through the service deployment manager at the Service Composition Layer.

### 3.4. Business Process Layer

Business Process Layer (BPL) basically represents the Business Process Layer in the existing architectures. As the proposed system uses BPM approach for providing an IoT application development environment, the layer has been termed as the BPL. At the BPL, the application specific components include the BPM Editor from [46], The Process Object Repository and the Process Execution Engine. The BPL utilizes the service objects composed at the service composition layer and represents them as business process modeling notations. Normally a service object is represented as a BPMN task notation while other notations such as Gateways are defined at the BPL for functions such as condition evaluation or setting multiple paths in a process model, Script notations represents data processing and generalized actions while Events represent the start and end of process models. Events can be further utilized for message passing among tasks and other notifications but the current implementation utilizes them only as the demarcating elements in the IoT process models.

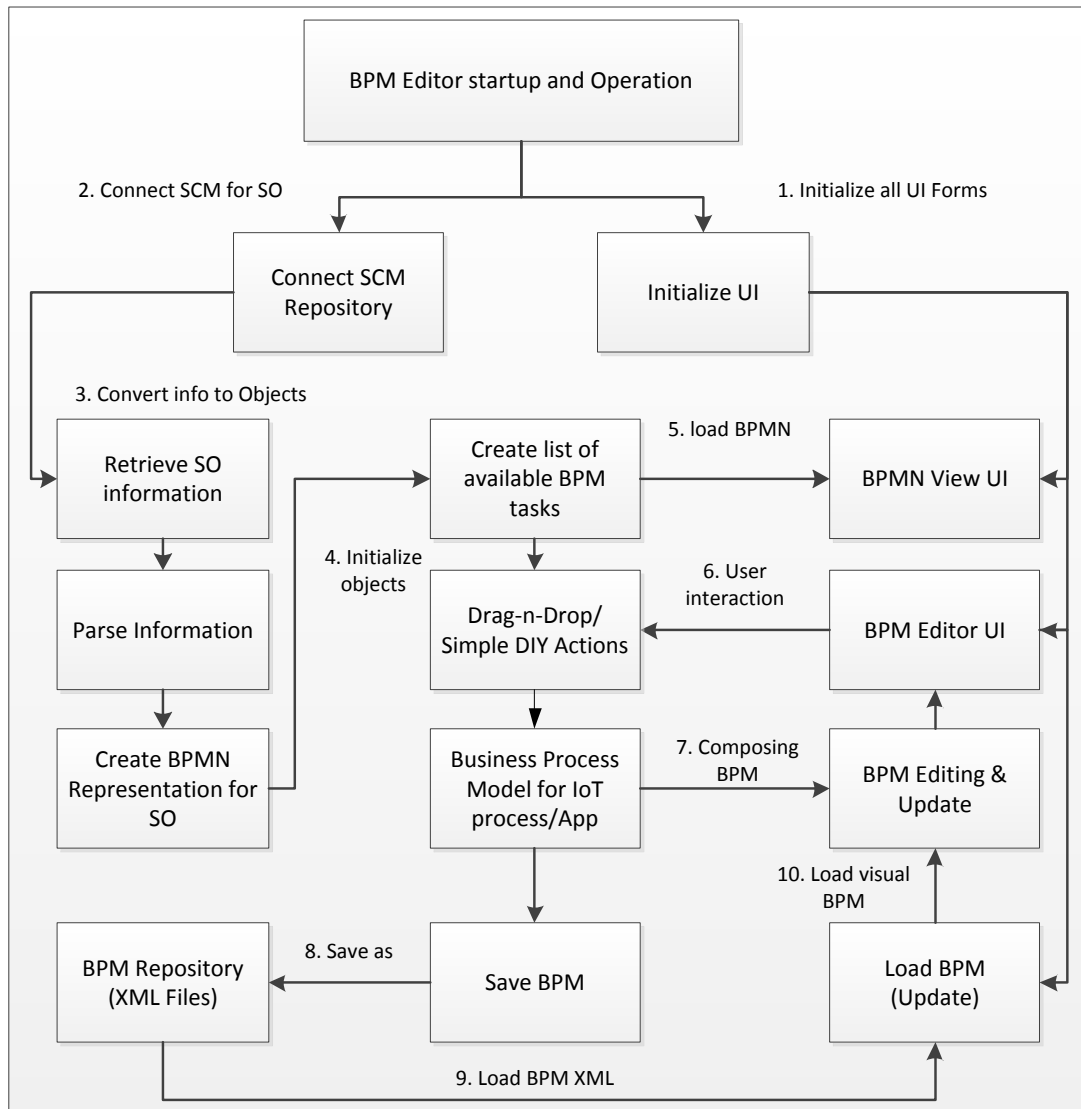
The BPM Editor application enables the user to create a functionality flow for an IoT environment in the form of a graphical business process model. The user creates rules and applies various scripts according to the conventions of the BPMN. This model can be saved as a process object in the repository at BPL and reloaded into the editor application for update at any time. An optimized version of the same file is created which can be used by the Deployment Manager at the BPL.

BPM Editor is the main component of the Business Process Layer (BPL) which acquires the XML representations of the Service Objects from the SO repository at the Service Composition Layer and represents them as Business Process Modeling Notation (BPMN) for the user. BPMN is a standardized set of notations used for requirements analysis in software development and for describing processes in business setups. The user utilizes the standardized notations with an intuitive drag-n-drop approach to design IoT processes or applications according to their own needs. Figure 20 shows the startup and operational configuration for the BPM.

The BPM Editor initializes all the user interface components and the communication components at the startup. In the figure, the communication interface is represented by the second step if the SO repository is located at a remote location otherwise a simple IO operation is performed to retrieve the XML files representing the service objects. The user interfaces of the BPM Editor include a BPMN Panel which displays the general notations for creating a business process flow. These general BPM notations include Task notation, Gateway notation, Script and event notations. The service objects retrieved from the repository are represented as BPMN tasks while the other notations provide supporting logic for the creation of BPM flows. These BPMNs are XAML based classes which can be dragged and dropped onto the main canvas by the user. The BPMN Panel is basically populated when the parser module associated with the BPM Editor receives and parses the service objects from SCM repository. The parser module parses the xml files, retrieves the input and output components of the service object along with the operational

rules if any, and initializes the BPM notations according to the tasks represented by the service objects.

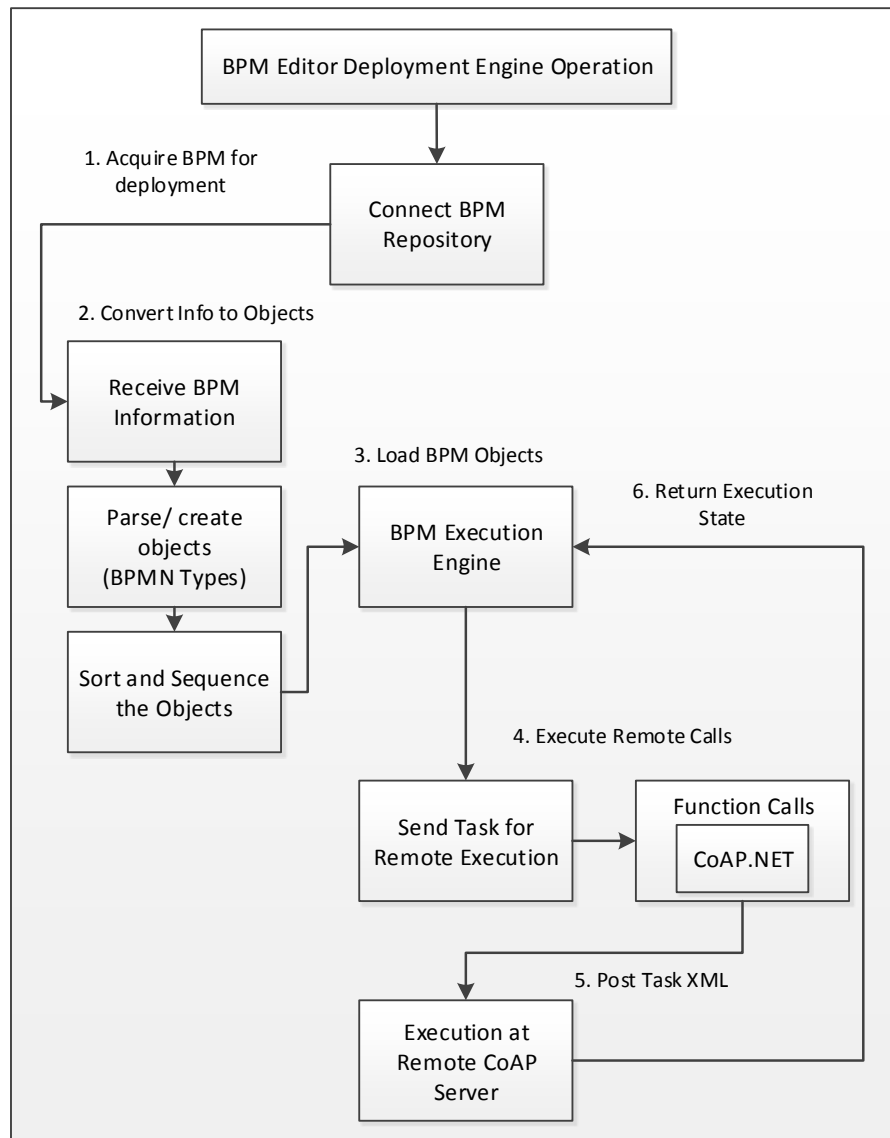
BPM Editor initializes all the user interface components and the communication components at the startup. In the Figure 20, the communication interface is represented by the second step if the SO repository is located at a remote location otherwise a simple IO operation is performed to retrieve the XML files representing the service objects. The user interfaces of the BPM Editor include a BPMN Panel which displays the general notations for creating a business process flow. These general BPM notations include Task notation, Gateway notation, Script and event notations.



**Figure 20: BPM Editor Operational Configuration**

The service objects retrieved from the repository are represented as BPMN tasks while the other notations provide supporting logic for the creation of BPM flows. These BPMNs are XAML based classes which can be dragged and dropped onto the main canvas by the user. The BPMN Panel is basically populated when the parser module associated with the BPM Editor receives and parses the service objects from SCM repository. The parser module parses the xml files, retrieves

the input and output components of the service object along with the operational rules if any, and initializes the BPM notations according to the tasks represented by the service objects.



**Figure 21: BPM Deployment Engine Operation Configuration**

The user then composes a BPM by using drag-n-drop and simple actions such as mouse-clicks and etc. The graphical BPM created at this stage basically represents the operational logic of the IoT process or IoT application. The BPM Editor UI provides editing functionalities such as copy, paste and delete etc. to provide an easy editing environment to the user. Once the BPM composition process is completed by the user, the graphical BPM is converted into an XML

representation for storage and later on for loading into the BPM Editor for further updates and changes if the user wishes so.

The BPM Deployment Engine is responsible for the deployment and execution of the BPM models created by the users through the BPM Editor. The BPM Deployment Engine creates a sequence for the execution of individual tasks by parsing the connections among various BPM notations as part of the graphical model along with the location of each graphical item in the model.

The user can then choose to deploy the model. For the execution of the deployed BPM, the execution engine is responsible to send the XML representation of each task to the concerned remote IoT resource. The execution state or any data is returned to the execution engine which further decides how to proceed with the execution of the BPM. The BPMN Gateways provides branching and execution logic for the process and the BPMN Scripts provide functions such as data processing or network communications which are too costly for the remote IoT resources and thus are executed by the execution engine.

In order to provide a detailed and concise picture of the whole system's design Figure 22– shows the overall collaboration among various layers with the BPL at the center of the process. The CoAP enabled IoT resources means the physical devices with some sort of CoAP services and functionality which can be invoked remotely via a CoAP client. For these IoT resources, the users create virtual objects by providing the necessary information to the system via the Virtual Object Manager interface. The virtual objects created are utilized by the Service Composition Manager to enable the users to compose Service Objects. The SCM implements simple and intuitive operations such as drag-n-drop and mouse-clicks to provide customized service object composition environment. A Service Object composed by the user consists of three main components i.e. Input resource, Output resource and the operational rule. The input resource is normally represented by the VO of a sensing device such as thermistor or gas detector etc., the



output resource is represented by a VO of an actuating device such a buzzer or LED and the operational condition is graphically represented by a join between the two resources.

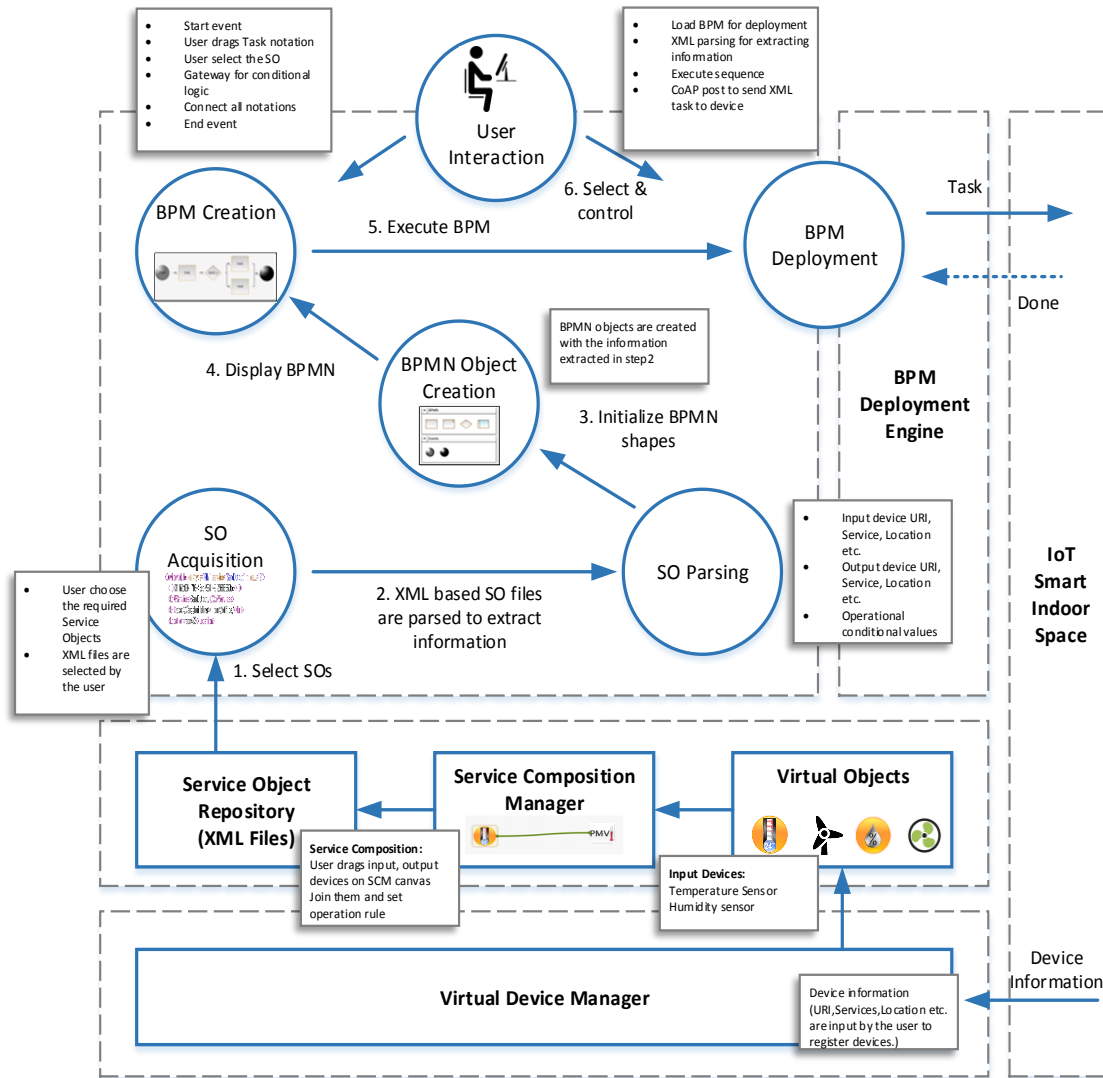


Figure 22: IoT Application Development Procedure in terms of the BPL Perspective

The join basically set the trigger condition for the actuating resource for the input resource. The graphically composed. Service Objects are editable by the user as an XML version of each SO is saved to the Service Object Repository at the Service Composition Layer. The same

repository is used by the Business Process Layer to acquire the definitions of Service Object in order to represent them as Business Process Modeling Notations.

For the conversion of Service Objects into Business Process Modeling Notations (BPMN), the BPM Editor first acquires the relevant Service Objects from the SO Repository at the Service Composition Layer. This has been represented as the first step in the collaboration diagram. The SO are in XML format and the BPM Editor uses the internal parser module to extract the necessary information about the associated resources. This information is used to initialize the BPMN objects which are clone-able graphical representations so that users can utilize them to create their BPM based IoT application model. BPM Editor provides a drag-n-drop based DIY modeling environment for the users to develop process models for their IoT applications and it also enables them to easily edit and change their models. The graphical models are also saved as XML files for this purpose to enable the users to share and update the models using the BPM Editor.

Once a model is completed and the user wants to deploy the IoT application represented by the BPM, the optimized XML version of the model is loaded into the BPM Deployment Engine. The BPM Deployment Engine is separate module at the Business Process Layer which is capable of communicating with the remote IoT devices. The model is parsed and converted into SO based tasks along with the application logic provided by the BPM notations. A sequence of execution is generated based on the graphical composition of the model as created by the user. The execution engine at the BPM Deployment Engine then uses the sequence to send XML based task definitions to the remote devices via CoAP Post calls. The remote devices executes the tasks by executing any relevant CoAP services.

### 3.5. Application Layer

The application layer specifies the communication protocol and interfaces used for the physical IoT prototype. This layer consists of three types of node: sensor node, proxy node and

actuator node. Sensor node is responsible for collecting environment data from physical IoT sensors. Actuator node takes charge of operating the physical IoT actuators. Proxy node is in charge of supporting communication between sensor nodes and actuator nodes. Logic objects provides intelligent prediction and control services to operate the IoT actuators. Communication channel defines the specified communication protocol for data and command transmission.

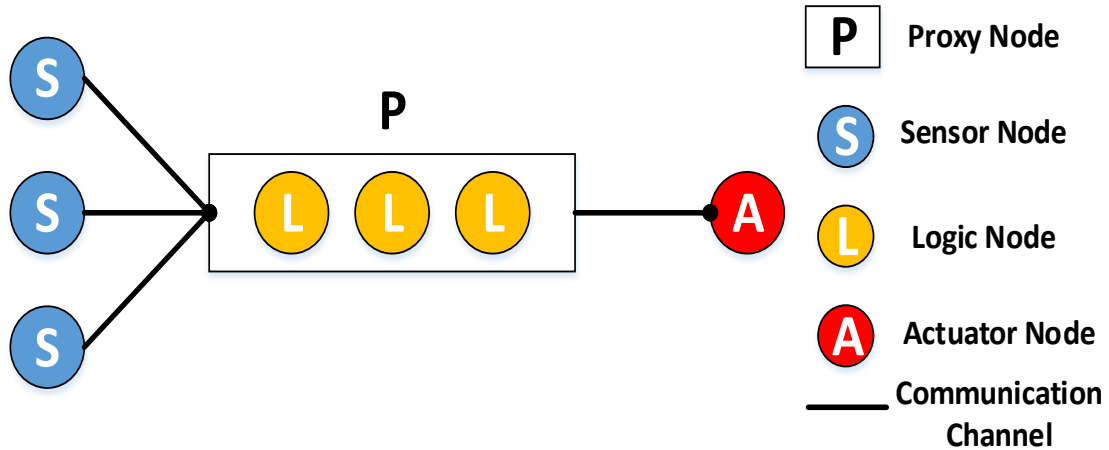


Figure 23: IoT Application Prototype Conceptual

Figure 24 illustrates the design of the proposed IoT application prototype. Three sensor nodes (Temperature, Humidity and Wind Speed) are connected with the proxy. The proxy contains three logic objects which provides the PMV calculation, prediction and fuzzy controlling functions in order to automatically control the fan actuator node in the indoor environment. The following section explains the functionality of logic objects in detail.

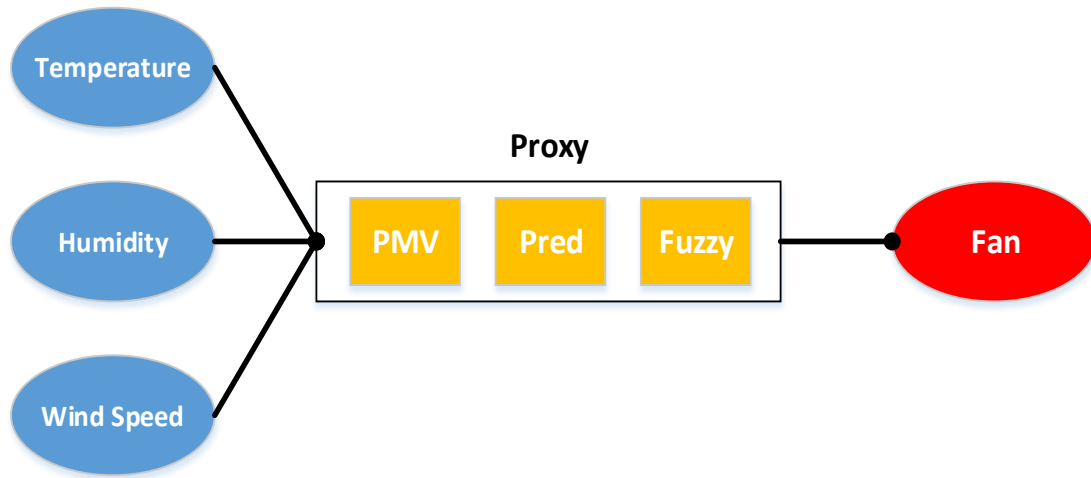


Figure 24: Design of the IoT Application Prototype

### 3.6. PMV (Predicted Mean Vote)

PMV [47] is arguably the most widely used thermal comfort index today. The ISO Standard 7730 (ISO 1984), "Moderate Thermal Environments - Determination of the PMV and PPD Indices and Specification of the Conditions for Thermal Comfort," uses limits on PMV as an explicit definition of the comfort zone. The Predicted Mean Vote (PMV) refers to a thermal scale that runs from Cold (-3) to Hot (+3) as shown as table 1. The original data was collected by subjecting a large number of people (reputedly many thousands of Israeli soldiers) to different conditions within a climate chamber and having them select a position on the scale the best described their comfort sensation. A mathematical model of the relationship between all the environmental and physiological factors considered was then derived from the data. The result relates the size thermal comfort factors to each other through heat balance principles and produces the following sensation scale.

**Table 1. Predicted Mean Vote Sensation Scale**

Predicted Mean Vote sensation scale

Value	Sensation
-3	Cold
-2	Cool
-1	Slightly cool
0	Neutral
1	Slightly warm
2	Warm
3	Hot

The PMV equation [48] only applies to humans exposed for a long period to constant conditions at a constant metabolic rate. Fanger's thermal comfort model is used to calculate PMV in the following equation.

(hc = convective transfer coefficient w/m<sup>2</sup> K)

$$H - 0.31(57.4 - 0.07H - Pa) - 0.42(H-58) - 0.0017M (58.7 - Pa) - 0.0014M (34 - Ta) = 3.9 \times 10^{-8} fcl \{ (Tcl + 273)^4 - (Tr + 273)^4 \} + fcl hc (Tcl - Ta) \quad (1)$$

Where the clothing surface temperature, Tcl, is given by

$$Tcl = 35.7 - 0.0275H + 0.155Iclo \{ H - 0.31(57.4 - 0.07H - Pa) - 0.42(H - 58) - 0.0017M (58.7 - Pa) - 0.0014M (34 - Ta) \}. \quad (2)$$

In addition to this, discomfort may occur when the skin is wetted (sweat, water, etc.) and Fanger has produced an equation for skin wetness which can be used as a test to exclude conditions which satisfy the comfort equation:

The problem with Fanger's equation is that when people are not satisfied, this is not a measure of how uncomfortable deviation is; therefore Fanger developed PMV = mean vote on ASHRAE scale (Hot warm slightly warm neutral slightly cool cold). PMV can be predicted from Fanger's equation thus:

$$PMV = 4 + (0.303 \exp(-0.036H) + 0.0275) \times \{6.57 + 0.46H + 0.31P_a + 0.0017HP_a + 0.0014HT_a - 4.13 f_{cl} (1 + 0.01dT) (T_{cl} - T_r) - hc_{cl} (T_{cl} - T_a)\}$$

where  $T_{cl}$  (surface temperature of clothed body) =  $35.7 - 0.0275H + 0.155 I_{clo} f_{cl} (4.13 (1 + 0.01dT) (T_{cl} - T_r) - hc_{cl} (T_{cl} - T_a))$   
 $hc_{cl} = 2.4(T_{cl} - T_a) (0.25 + 12.1 \sqrt{v})$  where  $v$  (air speed) which is greater and  $dT = T_r - T_a$ .

(3)

**Table 2. Nomenclature, description and measure units of the variables Involved in the PMV equation**

e	Euler's number (2.718)
$f_{cl}$	clothing factor
$h_c$	convective heat transfer coefficient
$I_{cl}$	clothing insulation [clo]
M	metabolic rate [ $W/m^2$ ] 115 for all scenarios
$P_a$	vapor pressure of air [kPa]
$R_{cl}$	clothing thermal insulation
$t_a$	air temperature [ $^{\circ}C$ ]
$t_{cl}$	surface temperature of clothing [ $^{\circ}C$ ]
$t_r$	mean radiant temperature [ $^{\circ}C$ ]
V	air velocity [m/s]
W	external work (assumed = 0)

Table 2 illustrates nomenclature, description and measure units of the variables Involved in the PMV equation.

The metabolic rate, or human body heat or power production, is often measured in the unit "Met". The metabolic rate of a relaxed seated person is one (1) Met, where

$$1 \text{ Met} = 58 \text{ W/m}^2 \text{ (356 Btu/hr)} \quad (4)$$

The mean surface area, the Du-Bois area, of the human body is approximately  $1.8 \text{ m}^2$  (19.4 ft<sup>2</sup>). The total metabolic heat for a mean body can be calculated by multiplying with the area. The total heat from a relaxed seated person with mean surface area would be

$$58 \text{ W/m}^2 \times 1.8 \text{ m}^2 = 104 \text{ W (356 Btu/hr)} \quad (5)$$

### 3.7. PMV Prediction based on Linear Regression Algorithm

In this section, we propose one linear regression-based algorithm to predict the PMV Index value in the indoor environment. The key idea is to adopt multiple linear regression (MLR) for estimating and evaluating environment parameter coefficients based on the recorded indoor sensing data. MLR is chosen because it adopts characteristic analysis, which attempts to model the environment parameter behavior for PMV Index value influence. First, we review briefly the multiple regression model [49]. There is a continuous random variable called the dependent variable,  $Y$  that stands for PMV Index value, and a number of independent variables,  $x_1, x_2, \dots, x_p$  which represents for temperature, humidity and wind speed. Our purpose is to predict the value of the dependent variable (also referred to as the response variable) using a linear function of the independent variables. The values of the independent variables (also referred to as predictor variables, regressors or covariates) are known quantities for purposes of prediction, the model is:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \varepsilon \quad (6)$$

where  $\varepsilon$ , the “noise” variable, is a normally distributed random variable with mean equal to zero and standard deviation  $\sigma$  whose value we do not know. We also do not know the values of the coefficients  $\beta_0, \beta_1, \beta_2, \dots, \beta_p$ . We estimate all these unknown values from the available data.

The data consist of  $n$  rows of observations also called cases, which give us values  $y_i, x_{i1}, x_{i2}, \dots, x_{ip}; i = 1, 2, \dots, n$ . The estimates for the  $\beta$  coefficients are computed so as to minimize the sum of squares of differences between the fitted (predicted) values at the observed values in the data. The sum of squared differences is given by

$$\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \dots - \beta_p x_{ip})^2 \quad (7)$$

Let us denote the values of the coefficients that minimize this expression by  $0, 1, 2, \dots, p$ . These are our estimates for the unknown values and are called OLS (ordinary least squares) estimates in the literature. Once we have computed the estimates  $0, 1, 2, \dots, p$ . We can calculate an unbiased estimate for  $\sigma^2$  using the formula:

$$\begin{aligned} \hat{\sigma}^2 &= \frac{1}{n - p - 1} \sum_{i=1}^n (y_i - \hat{\beta}_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \dots - \beta_p x_{ip})^2 \\ &= \frac{\text{Sum of the residuals}}{\# \text{observations} - \# \text{coefficients}} \end{aligned} \quad (8)$$

We plug in the values of  $0, 1, 2, \dots, p$  in the linear regression model (1) to predict the value of the dependent value from known values of the independent values,  $x_1, x_2, \dots, x_p$ . The predicted value,  $\hat{Y}$ , is computed from the equation

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_p x_p. \quad (9)$$



Predictions based on this equation are the best predictions possible in the sense that they will be unbiased (equal to the true values on the average) and will have the smallest expected squared error.

An important and interesting fact for our purposes is that even if we drop the assumption of normality and allow the noise variables to follow arbitrary distributions, these estimates are very good for prediction. We can show that predictions based on these estimates are the best linear predictions in that they minimize the expected squared error. In other words, amongst all linear models, as defined by equation above, the model using the least squares estimates.

$$\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \dots, \hat{\beta}_p, \quad (10)$$

will give the smallest value of squared error on the average. We elaborate on this idea in the next section.

The Normal distribution assumption was required to derive confidence intervals for predictions. In data mining applications we have two distinct sets of data: the training data set and the validation data set that are both representative of the relationship between the dependent and independent variables. The training data is used to estimate the regression coefficients  $0, 1, 2, \dots, p$ . The validation data set constitutes a “hold-out” sample and is not used in computing the coefficient estimates. This enables us to estimate the error in our predictions without having to assume that the noise variables follow the Normal distribution. We use the training data to fit the model and to estimate the coefficients. These coefficient estimates are used to make predictions for each case in the validation data. The prediction for each case is then compared to value of the dependent variable that was actually observed in the validation data. The average of the square of this error enables us to compare different models and to assess the accuracy of the model in making predictions.

### 3.8. Fan Control based on Fuzzy Logic with PMV

In a thermal control scenario, various standard control schemes, such as an on/off switching thermostats, proportional-integral (PI) and proportional-integral-derivative (PID), have been extensively used in building engineering [50], [51]. Generally all these schemes do not have any direct knowledge of the system to be controlled and they are designed with constant parameters. They thus provide poor control performance for noisy, disturbed and non-linear processes without taking into account users' behavior [52].

Although some PMV based control methods exist, they treat all occupants the same, do not take into account location and PMV itself [53] and are mostly model based approaches [54]. We developed a novel fuzzy controller for HVAC systems which takes into account the PMV index. Furthermore, considering all the control systems related issues we developed a model-free based controller and, at the same time, the adoption of fuzzy logic makes it easier to implement on a microcontroller.

The approach considered for the PMV optimization based control is the linguistic fuzzy modeling (LFM) with Mamdani rule structure [55] due to its capability to model human knowledge in an explicit way. Input variables chosen for the FIS control are:

- The value of the PMV index
- The variation of the PMV index

The output of the FIS engine is the FanCoilSpeed. The membership functions of the variables involved in the fuzzy system consist of triangular and trapezoidal functions chosen by experts in the field of thermal regulation. The membership functions used for the input and output fuzzy sets are shown Figure 25, 26, 27. The triangular membership functions are used for all the fuzzy sets of the input and the output vector. Mathematically, this representation can be interpreted as follows:

$$f(x; a, b, c) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & x \geq c \end{cases} \quad (11)$$

Values chosen for the input and output variables fuzzy sets are reported in Table 3.

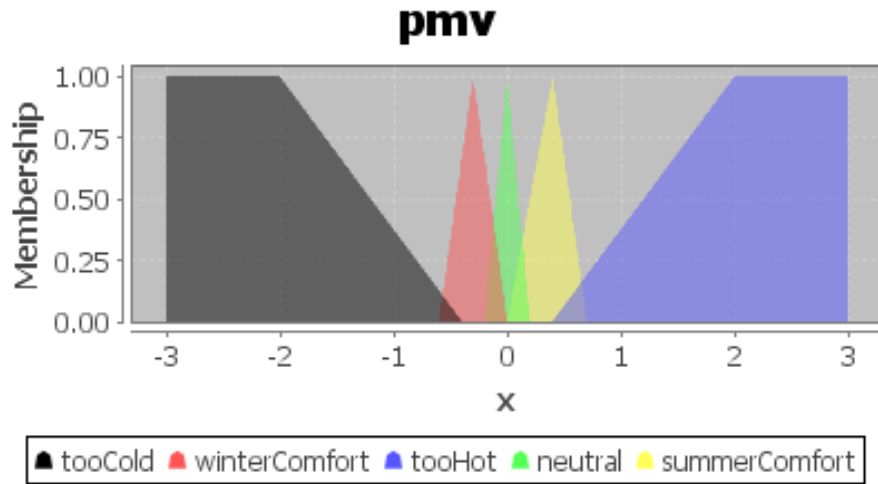


Figure 25: Fuzzy Sets for the Input Variable PMV

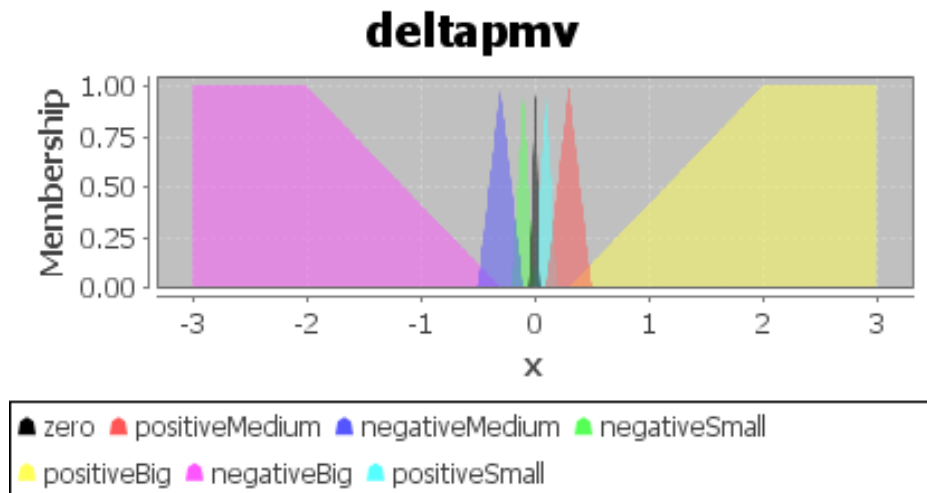


Figure 26: Fuzzy Sets for the Input Variable  $\Delta$ PMV

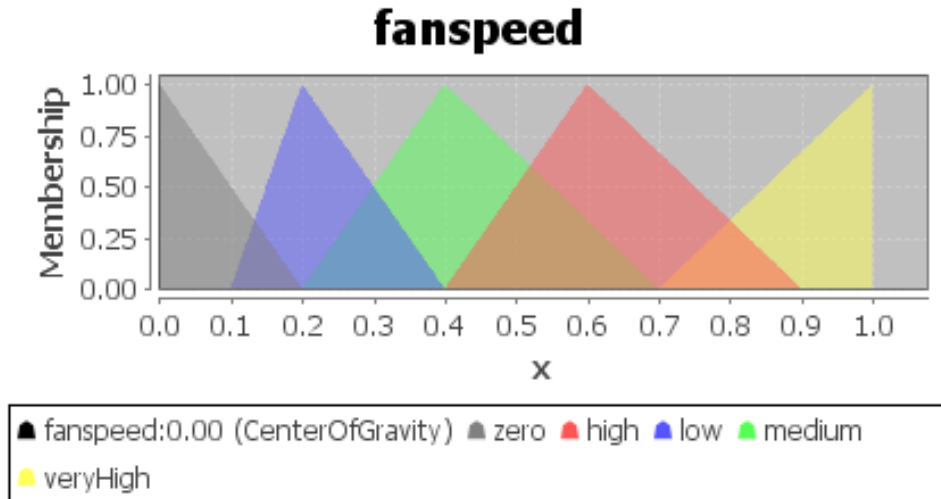


Figure 27: Fuzzy Sets for the Output Variable Fan Speed

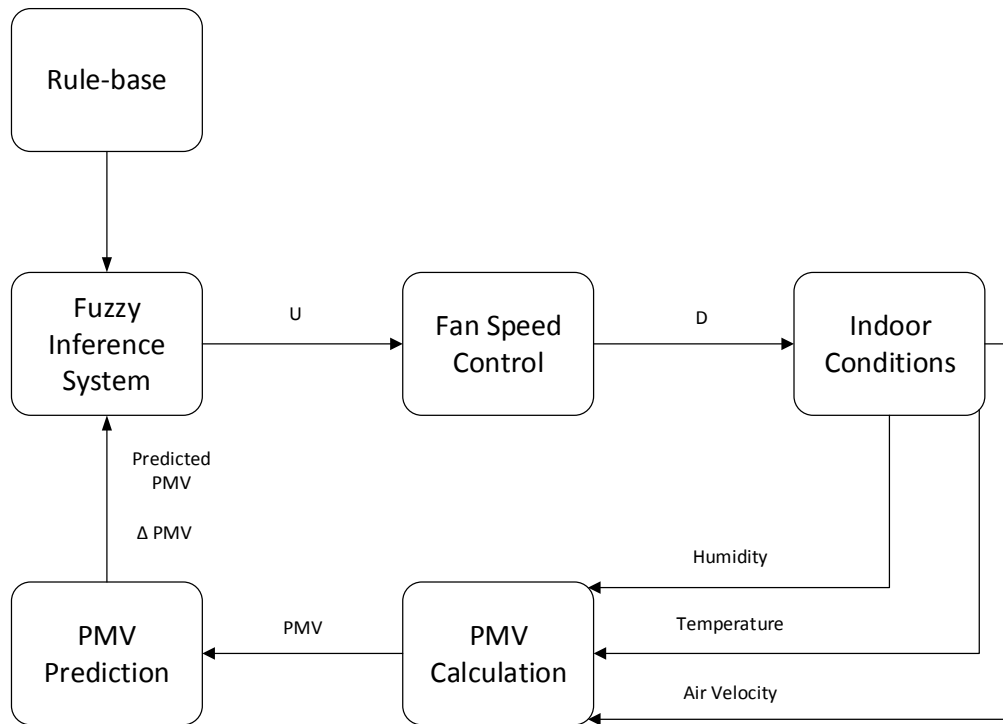
The Max-Min-mCoA fuzzy inference algorithm is considered. The fuzzy logic controller uses the following equation to calculate the geometric center of the full area under the scaled membership functions:

$$\text{Center of gravity} = \frac{\int_a^b \mu_A(x) x dx}{\int_a^b \mu_A(x) dx} \quad (12)$$

Where CoA is the modified center of area and  $f(x)$  the output of the inference process. The interval of integration is between the minimum membership function value and the maximum membership function value. Note that this interval might extend beyond the range of the output variable. The PMV method, as it was developed, basically states that the indoor temperature should not change according to the seasons considering one set temperature year-round. The aim of the fuzzy logic controller is to overcome this issue adapting the comfort conditions to the seasonality.

Table 3: Considered fuzzy sets for input and output variables

Input variables	Linguistic terms	Fuzzy sets (a,b,c)
Predicted Mean Vote (PMV)	Too Cold	-3, -2, -0.4
	Winter Comfort	-0.6, -0.3, 0
	Neutral	-0.2, 0, 0.2
	Summer Comfort	0, 0.4, 0.7
	Too Hot	0.4, 2 +3
PMV Variation ( $\Delta$ PMV)	Negative Big	-3, -2, -0.3
	Negative Medium	-0.5, -0.3, -0.1
	Negative Small	-0.2, -0.1, 0
	Zero	-0.05, 0.01, 0.05
	Positive Small	0, 0.1, 0.2
	Positive Medium	0.1, 0.3, 0.5
	Positive Big	0.3, 2, +3
Output variable	Linguistic terms	Fuzzy sets (a,b,c)
Fan Speed (U)	Zero	0, 0.2, 0.2
	Low	0.1, 0.2, 0.4
	Medium	0.2, 0.4, 0.7
	High	0.4, 0.6, 0.9
	Very High	0.7, 1, 1



**Figure 28. Fuzzy Controller Block Diagram**

Figure 28 depicts the block diagram of the comfort control process. PMV index is computed in the PMV calculation module and measuring indoor air temperature, relative humidity and air velocity. The current PMV index values are used to build prediction module to predict the PMV index. Values of predicted PMV and its variation are the inputs for the Fuzzy Inference System (FIS). The FIS output values (U) are transformed in the fan speed control module in which the control value (D) for the fan coil is its speed percentage (0 - 100%) with respect the max speed. Considering fuzzy sets of table 4, if the indoor comfort conditions is toohot then the control system acts to speed up the fan module to obtain and maintain the indoor environment comfort. On the contrary when the indoor comfort condition is neutral and the controller stops the fan module since the indoor environment condition is comfortable. Furthermore the PMV variation in the FIS allow the control system to act as a regulator, we built 35 fuzzy rules and a sample is shown in Table 4.

**Table 4: Sample of the fuzzy rules used**

<b>PMV</b>	<b><math>\Delta</math>PMV</b>	<b>U</b>
TH	NS	VH
TH	NB	H
SC	NM	Z
SC	PS	M
N	NS	Z
N	Z	Z

## 4. Implementation

Figure 29 represents the previously described architecture from the perspective of the use-case by highlighting the prototype IoT smart space. For the purpose of the smart space prototype demonstration, the temperature sensor, humidity sensor, wind sensor, proxy and the fan actuator are implemented as IoT devices. The Physical Network Layer of the architecture represents these components as separate physical devices. The information of the implemented device components is provided to the Virtual Object Layer which converts the information into virtual objects. The Service Logic Layer utilizes the virtual objects from the virtual object repository at the Virtual Object Layer and the logic objects from the service object repository at the Service Logic Layer. The virtual objects are visualized using icons which can be interacted with like any Windows based control. The Business Process Layer acquires the service object definitions from the SO repository at Service Layer, parses the XML representation of the service objects to extract information and then represent the services as BPMN task notations. Once the BPM is created, it can be deployed via the deployment engine and the IoT devices will perform the operations accordingly.

### 4.1. Virtual Device Manager

Figure 30 shows the screen shot of the Virtual Device Manager module at the Virtual Object Network Layer. This module helps the users to encapsulate the attributes of their IoT resources. It provides an intuitive way to bind the IoT resources with a virtual representation so that inexperienced users are able to interact and manipulate. User manually provides the details with respect to their IoT resources in the form of URI, location, type and properties. Any device which



supports the proposed protocols can be added as a resource to the system via this approach. URI specifies the protocol and address through which the device can be uniquely identified. Device type tag specifies the type of the specific IoT devices. Location tag is used for specifying the location of the remote IoT resource. Properties tag specifies the available services which can be executed via the specific resource.

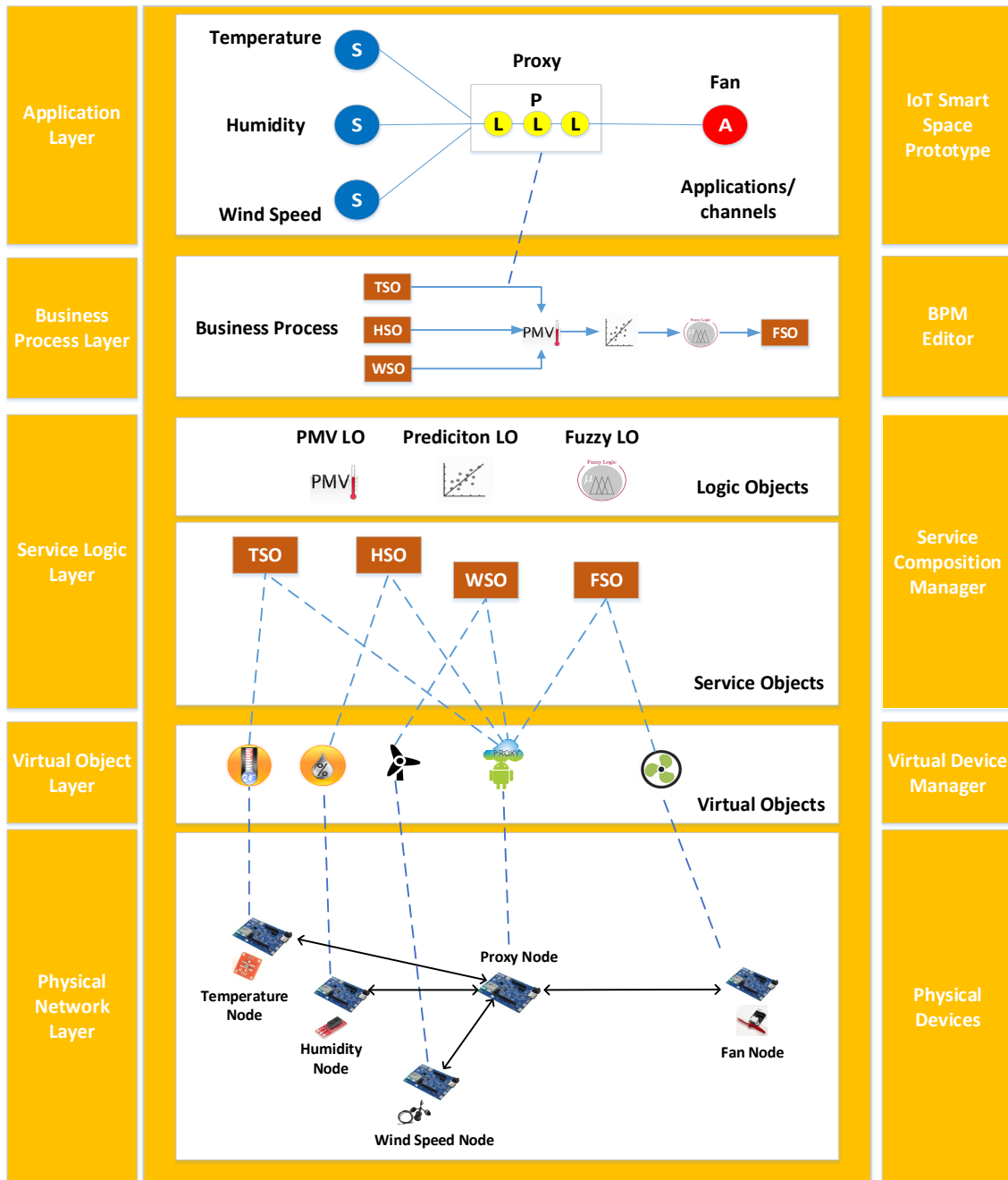
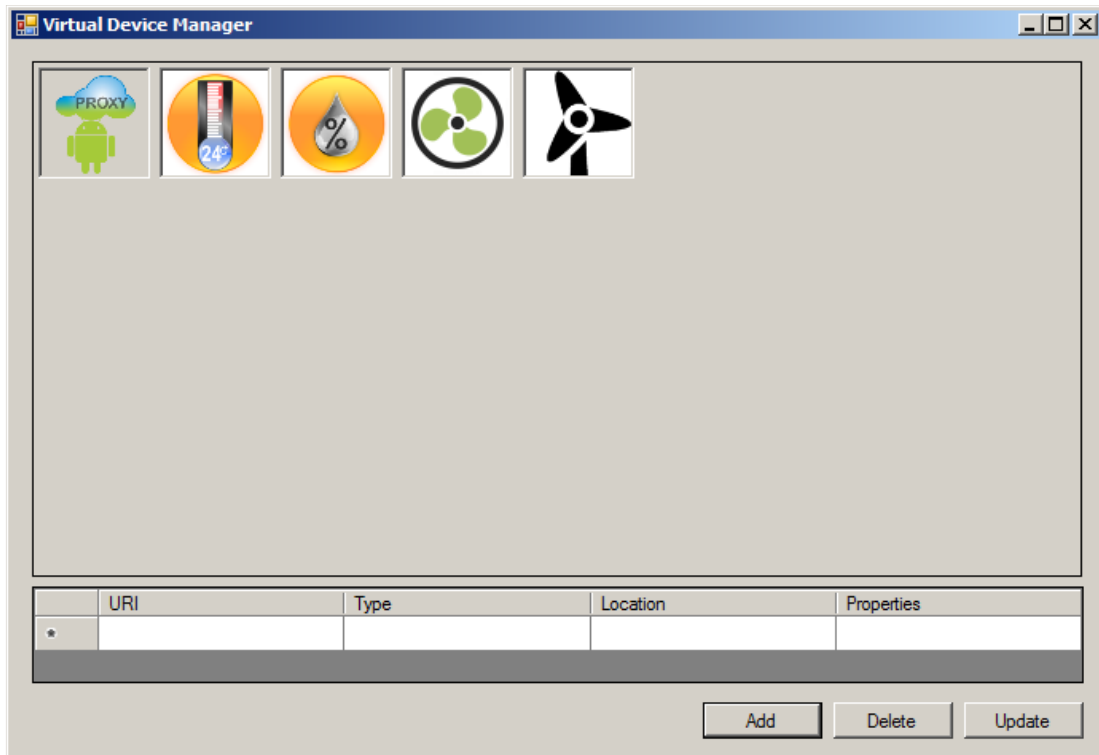


Figure 29: IoT Smart Space Implementation in the Proposed Architecture



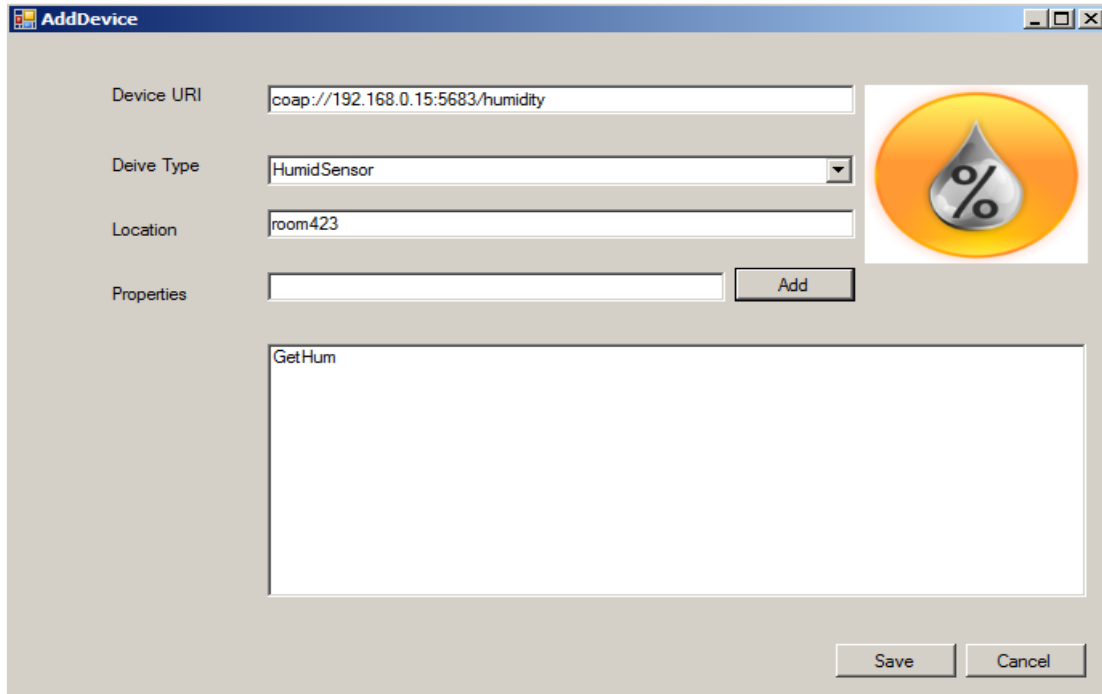
**Figure 30: Virtual Device Manager Main Interface**

Figure 31 represents the screen shot for creating virtual device in the Virtual Device Manager. Users need to specify the device URI, device type, location and properties. After that, users can save the input information and the Virtual Device Manager will automatically generate the virtual device information using XML representation as shown in Figure 31.

Table 5 illustrates device in the smart indoor space which are temperature sensor module, humidity sensor module, wind sensor module, fan module and proxy module.

Temperature sensor module: TinkerKit T000200 thermistor is specifically designed to be used with the TinkerKit development toolkit for Arduino. The Thermistor is a resistor whose resistance varies significantly (more than in standard resistors) with temperature. This module's output approaches 5v as the temperature increases. As the temperature decreases, it approaches 0V. The module has been utilized to implement Intel Edison based CoAP services for providing temperature values in Centigrade as well as Fahrenheit scales.

Wind sensor module: Wind sensor is used to measure the wind speed. The SEN1070 is a three-cup anemometer that monitors wind speed for the range of 0 to 30m/s. The wind sensor module has been utilized to implement Intel Edison based CoAP services for providing the air speed values by level.



**Figure 31: Device Interface for Creating Virtual Objects**

Humidity sensor module: The HIH-4030/4031 Series Humidity Sensors are designed specifically for high volume OEM (Original Equipment Manufacturer) users. Direct input to a controller or other device is made possible by this sensor's near linear voltage output. With a typical current draw of only 200  $\mu$ A, the HIH-4030/4031 Series is often ideally suited for low drain, battery operated systems. The humidity sensor module has been utilized to implement Intel Edison based CoAP services for measuring humidity values.

Fan module: Arduino L9110 module is used as an IoT actuator in the proposed prototype. L9110 drive which can control the positive & negative turning with mounting holes, high quality and high efficiency. The fan actuator module has been utilized to implement Intel Edison based CoAP services for setting fan coil speed.

IoT proxy module: The IoT proxy establishes the connection between the virtual domain and physical domain. The IoT proxy module has been utilized to implement Intel Edison based CoAP / IoTivity services for transferring sensing data from physical domain to virtual domain and providing prediction service to maintain indoor comfort index.

**Table 5: Device Implementation Summary for Smart Indoor Space**

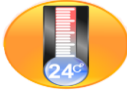




Devices	Temperature Sensor	Wind Sensor	Humidity Sensor	Fan Actuator	Proxy Node
Visual Representation					
Servo Model	TinkerKit T000200 thermistor module	Wind Speed Sensor (SKU:SEN0170)	Humidity Sensor (HIH-4030)	Arduino L9110 Fan Module	None
Server	Intel Edison with libcoap	Intel Edison with libcoap	Intel Edison with libcoap	Intel Edison with IoTivity	Intel Edison with CoAP/ IoTivity
Function	GetTempC, GetTempF	GetAir	GetHumidity	SetFan	GetPMV, MLR Fuzzy

Figure 32 presents the XML representation for virtual objects in the form of device nodes. Four nodes have been expanded in the figure which shows the stored information for a temperature sensor, humidity sensor, wind sensor and a fan device. The URI specifies the protocol and address through which the device can be uniquely identified. The Properties tag specifies the available services which can be executed via the specific resource. A resource can also have more than one

property (executable functions) which is specified by the sub-tags <P> in the Properties tag. Both URI and an instance of the Properties tag can be utilized to provide a uniquely addressable function of the remote resource.

The Location tag is used for specifying the location of the remote IoT resource. This tag and more information regarding the owner or allowed users etc. can be considered for future studies related to the security of the system.

```

<?xml version="1.0" encoding="UTF-8"?>
- <Devices>
  - <Device>
    <Uri>coap://192.168.0.14:5683/temperature</Uri>
    <Type>TempSensor</Type>
    <Location>room423</Location>
    - <Properties>
      <P>GetTemp</P>
    </Properties>
  </Device>
  - <Device>
    <Uri>coap://192.168.0.15:5683/humidity</Uri>
    <Type>HumidSensor</Type>
    <Location>room423</Location>
    - <Properties>
      <P>GetHum</P>
    </Properties>
  </Device>
  - <Device>
    <Uri>coap://192.168.0.10:5683/fan</Uri>
    <Type>Fan</Type>
    <Location>room423</Location>
    - <Properties>
      <P>SetFan</P>
    </Properties>
  </Device>
  - <Device>
    <Uri>coap://192.168.0.11:5683/windspeed</Uri>
    <Type>WindSensor</Type>
    <Location>room423</Location>
    - <Properties>
      <P>GetWind</P>
    </Properties>
  </Device>

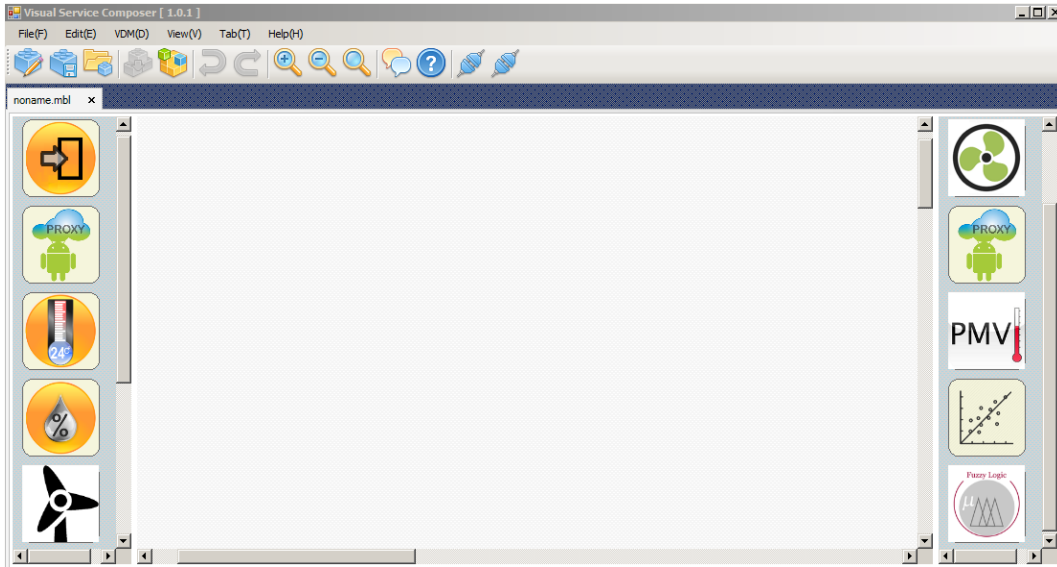
```

Figure 32: XML Representation of Created Virtual Devices

## 4.2. Service Composition Manager

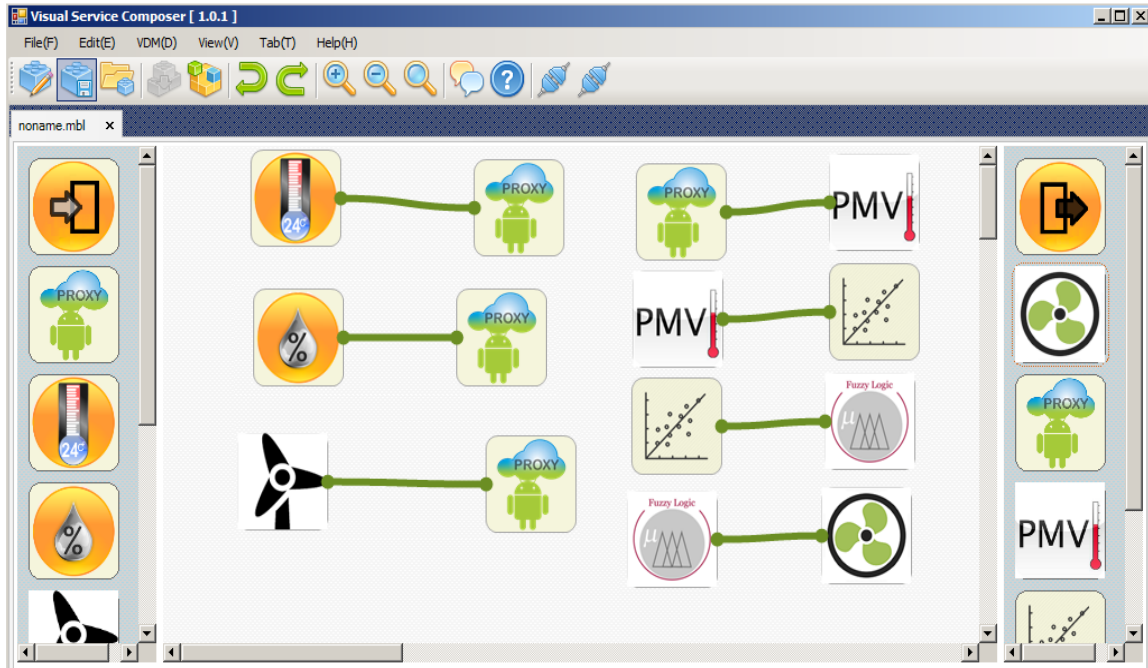
Service Composition Manager (SCM) is the main module at the Service Layer. A snap shot of the SCM interface is shown in Figure 33. The main objective of this module is to allow the user to easily visualize, interact with and manipulate the virtual objects created by Virtual Device Manager. The SCM is developed in C# environment as a Windows form application.

For this purpose the virtual objects are separately represented as input and output modules. These modules can be directly dragged and dropped on a canvas through the basic Windows OS mouse events. The VO modules on the canvas can then be connected with simple joining lines which represents connection between the input and output VOs. Finally, the user can set the rules of operations for the joined virtual objects. For this purpose simple and intuitive approach has been implemented. The user can double-click each VO to display the settings form for the VO. There the user can specify the values of attributes, set ranges for the conditional operations and choose conditional operators for the evaluation of conditional logic. The process is illustrated in Figure 33.



**Figure 33: Service Composition Manager Interface**

The SCM interface provides user-centric approach for the development of service objects based on the virtual representation of IoT resources. The interface is implemented with standard tool strip for efficient editing and composition of service objects to enable user efficiency and better DIY environment. Menus and shortcuts have been implemented for providing the user with standardized editing and composition functions such as cut, copy, paste, detailed viewing of the graphical models and commenting the models for easy recognition of implemented functionalities.



**Figure 34: Service Object Process in Service Composition Manager**

The joining of input and output VOs in the SCM creates a service object (SO). The process has been illustrated in Figure 34. These SOs are stored as XML documents which separately represent each input and output VO as part of a service object. The connections between these VOs are also represented in the XML document in the form of a join node which specifies the source and sink entities for the connection. Hence the VOs can be stored, opened and updated according to the user requirements.



**Table 6: Logic Objects Implementation Summary**


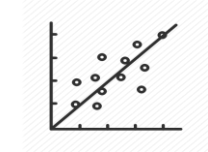

<b>Objects</b>	<b>PMV Index</b>	<b>LR Predictor</b>	<b>Fuzzy Controller</b>
<b>Visual Representation</b>			
<b>Server</b>	Intel Edison with Californium	Intel Edison with Californium	Intel Edison with Californium
<b>Function</b>	GetPMV	StartPrediction	StartFIS
<b>Library</b>	Native Java	Weka	jFuzzyLogic

Table 6 illustrates the detail information of the logic objects at the Service Layer. The PMV Index module is designed to compute PMV index value using sensing data from the other IoT sensors. The module has been utilized to implement Intel Edison based CoAP services in pure Java programming language. The LR Predictor module is specifically designed to provide PMV index prediction service based on sensing data information. The module has been utilized to implement Intel Edison based CoAP services using Weka library. The Fuzzy Controller module is specifically designed to control the IoT fan actuator considering PMV index value and PMV index variation value. The module has been utilized to implement Intel Edison based CoAP services using jFuzzyLogic library.

Figure 35 shows a sample of the XML documents representing Service Objects (SO). Each SO is represented by the JoinInfo tag where unique identifiers specifies the input and output device associated with the specific service object. The same identifiers are used in the DeviceModule tags as shown in the figure. The DeviceModule tags encapsulate the information about each resource as part of the saved service objects. This information include the device type, complete URI to

access the remote IoT resource, the service name selected by the user at SCM to be executed along with the operational conditions as part of the SO for the specific device and the location of the remote resource.

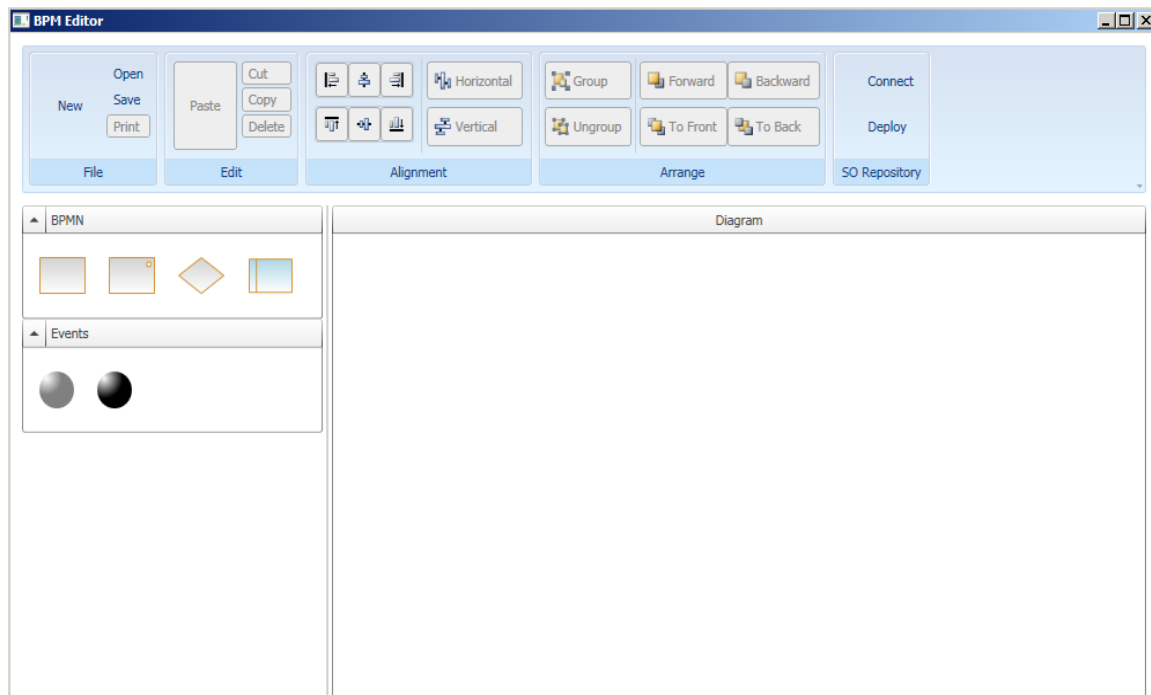
```
<?xml version="1.0" encoding="utf-8"?>
<Workspace xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Devices>
    <DeviceModule xsi:type="Temperature" temp="20" service="GetTempC" range_T="over">
      <ID>5fff91d5-c49a-4df1-9335-29baf0dbef0d</ID>
      <CoAPServices>GetTemp</CoAPServices>
      <Uri>coap://coputer-engg/office/</Uri>
      <Location>room423</Location>
      <Position>
        <X>247</X>
        <Y>149</Y>
      </Position>
    </DeviceModule>
    <DeviceModule xsi:type="Fan" fanlevel="2" service="SetSpeed" time_L="1">
      <ID>0abe6159-e3c4-48ea-b80d-5adb40198f45</ID>
      <CoAPServices>SetSpeed,</CoAPServices>
      <Uri>coap://enggbuilding4/floor4/office/</Uri>
      <Location>room423</Location>
      <Position>
        <X>641</X>
        <Y>198</Y>
      </Position>
    </DeviceModule>
  </Devices>
  <Joins>
    <JoinInfo InputType="5fff91d5-c49a-4df1-9335-29baf0dbef0d" OutputType="0abe6159-e3c4-48ea-b80d-5adb40198f45" />
  </Joins>
  <FilePath>C:\Users\Hanglei\Documents\SCDesigner</FilePath>
  <FileName>SOexample.mbl</FileName>
</Workspace>
```

Figure 35: XML Representation of Service Objects

The list of names encapsulated in the tag named CoAPServices represents all the services supported by the specific device. This list is included in the service object definitions for enabling the SCM to de-serialize the XML files and graphically render it with complete information if the user wishes to update the SO later. The location information would further be utilized for security and user rights allocation in the future studies.

### 4.3. BPM Editor

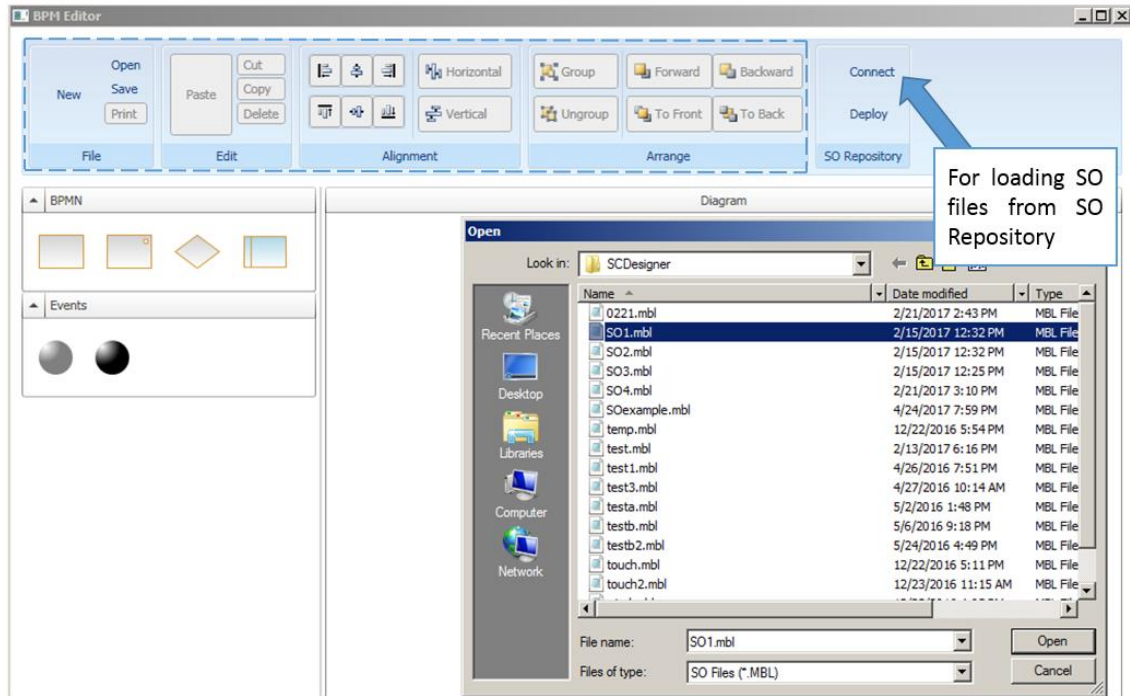
Business Process Model Editor is responsible for the representation of the Service Objects composed at the SCM as Business Process Modeling Notations (BPMN). The main interface for BPM Editor is shown in Figure 36. The aim of providing a BPMN based representation of the Service Objects is to provide a DIY interface for anyone with the basic knowledge of the notations to create and deploy their IoT applications. It also eliminates the requirement of any programming skills because the user just has to create a graphical model and it is directly deployed as an IoT application.



**Figure 36: BPM Editor Interface**

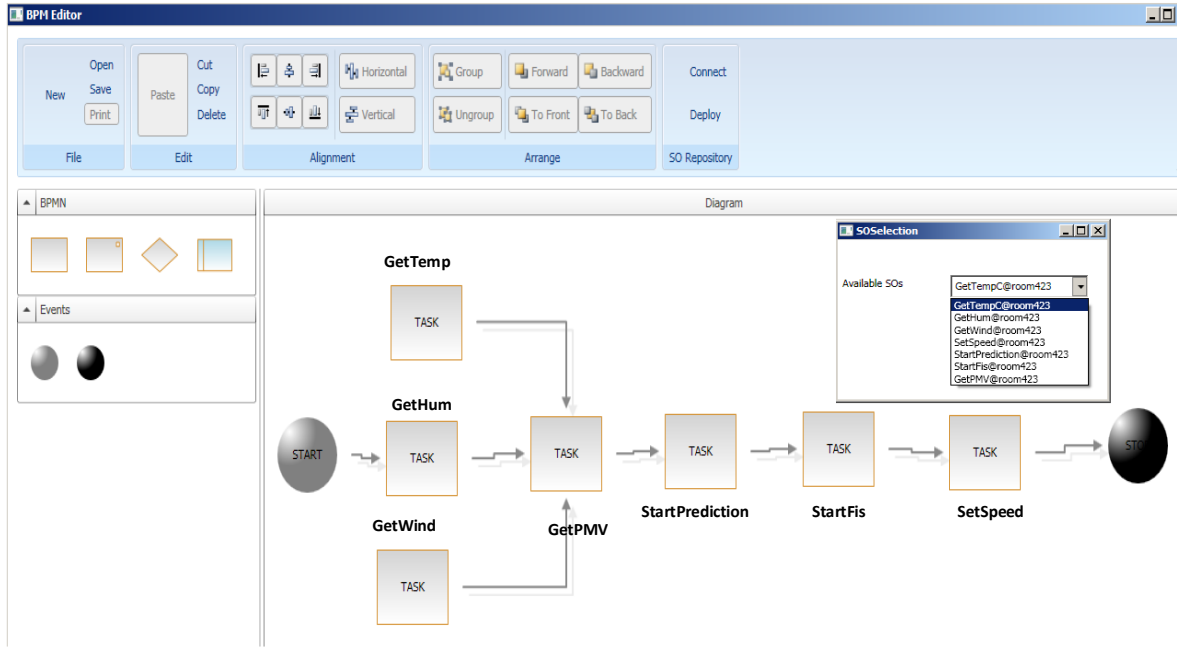
The main interface is divided into three main areas. The first is the BPMN Panel on the left side. This panel groups the various implemented notations in the form of a shape palette. Each shape in the BPMN panel is an instance of the XAML based class which is derived from a common class termed as `ToolboxItem`. This class provides the cloning attributes to each shape derived from it and thus enabling the shapes to be dragged and dropped by the users. The main BPMN notations

which have been implemented as part of this prototype system include the Task, Script, Gateway and the swimlane notation. In the event section of the BPMN panel, Start and Stop events have been implemented. These event notations specify the start and finish of a process represented by the graphical BPM.



**Figure 37: Loading Service Objects at BPM Editor**

The top area of the interface provides an application level toolbar containing editing functionalities necessary for shapes and model composition. The toolbar is implemented in a standardized way to resemble the toolbars provided by well-known and common applications such as Microsoft Word and Microsoft PowerPoint. This feature enables the users to easily recognize various editing functionalities through general knowledge and helps in model development.



**Figure 38: Generated BPMN in BPM Editor**

For a new BPM project to be created by the user, the user must import the necessary Service Objects that will be part of the specific BPM. For this purpose, the BPM Editor provides a simple interface where the user can connect to the SO repository at the Service Composition Layer and select the XML files for the necessary SOs as shown in Figure 37. The SOs are read by the FileManager and parsed by the XMLParser to extract the SO components and attributes. A list of the basic descriptions for the service objects is created which is associated with the BPMN task shape when the user double-clicks it in the editor canvas area. Service objects are represented as Tasks and the user can create their model via the same drag-n-drop approach. The sequence of operation is created by connecting the notations via arrow objects and the same arrow objects are utilized to capture information regarding the inputs and outputs of notations in the model. As mentioned earlier, the conditional logic of the process is implemented using the BPMN gateway notations while processor intensive tasks and remote communication tasks which are not suitable to be executed on the remote IoT resources are represented by the Script notations. The Script

notation has been provided with a list of scripts from which the user can choose to manipulate or process the data. The process of BPM creation by the user is illustrated in Figure 38.

The BPM model created by the users via the BPM Editor is stored as an XML file. This file is the direct serialization of graphical notation and the associated information such as location on the canvas, identifiers etc. If the user chooses to reload a previously created BPM into the Editor for updates or changes, the file contains all the information to enable the BPM Editor to load and re-render the same graphical model as created by the user.

```

<DesignerItem>
  <Left>109</Left>
  <ID>f9c503d8-6f94-4db3-ba8d-437e9c036d5d</ID>
  <ItemType>TASK</ItemType>
  <DevType>Temperature</DevType>
  <SoUri>coap://192.168.0.21:5683/temperature/</SoUri>
  <SoName>GetTempC</SoName>
  <SoLocation>room423</SoLocation>
  <Condition_Op>equal</Condition_Op>
  <Condition_val>0</Condition_val>
  <zIndex>1</zIndex>
  <IsGroup>>false</IsGroup>
  <ParentID>00000000-0000-0000-0000-000000000000</ParentID>
</DesignerItem>

```

**Figure 39: Temperature Sensor XML Representation of BPMN**

```

<DesignerItem>
  <Left>230</Left>
  <ID>616790be-9714-4e5a-832c-5c273e97af2f</ID>
  <ItemType>TASK</ItemType>
  <DevType>PMV</DevType>
  <SoUri>coap://coap://192.168.0.5:5683/pmv/</SoUri>
  <SoName>GetPMV</SoName>
  <SoLocation>room423</SoLocation>
  <Condition_Op></Condition_Op>
  <Condition_val>2</Condition_val>
  <zIndex>7</zIndex>
  <IsGroup>>false</IsGroup>
  <ParentID>00000000-0000-0000-0000-000000000000</ParentID>
</DesignerItem>

```

**Figure 40: PMV Index XML Representation of BPMN**

```

<DesignerItem>
  <Left>378</Left>
  <ID>0114d4b6-db35-437b-94b9-78b3a008eb20</ID>
  <ItemType>TASK</ItemType>
  <DevType>LROutput</DevType>
  <SoUri>coap://coap://192.168.0.5:5683/LrPred/</SoUri>
  <SoName>Start Prediction</SoName>
  <SoLocation>room423</SoLocation>
  <Condition_Op></Condition_Op>
  <Condition_val>2</Condition_val>
  <zIndex>9</zIndex>
  <IsGroup>>false</IsGroup>
  <Parent ID>00000000-0000-0000-0000-000000000000</Parent ID>
</DesignerItem>

```

Figure 41: LR Predictor XML Representation of BPMN

```

<DesignerItem>
  <Left>372</Left>
  <ID>c339153a-00f6-4a0d-bd62-bd3860e033f9</ID>
  <ItemType>TASK</ItemType>
  <DevType>FuzzyOutput</DevType>
  <SoUri>coap://coap://192.168.0.5:5683/FisControl/</SoUri>
  <SoName>Start Fis</SoName>
  <SoLocation>room423</SoLocation>
  <Condition_Op></Condition_Op>
  <Condition_val>2</Condition_val>
  <zIndex>11</zIndex>
  <IsGroup>>false</IsGroup>
  <Parent ID>00000000-0000-0000-0000-000000000000</Parent ID>
</DesignerItem>

```

Figure 42: Fuzzy Controller XML Representation of BPMN

```

<DesignerItem>
  <Left>508</Left>
  <ID>3a034303-d6b7-4ac7-996e-c49e934d47ae</ID>
  <ItemType>TASK</ItemType>
  <DevType>Fan</DevType>
  <SoUri>coap://192.168.0.10:5683/Set Fan/</SoUri>
  <SoName>Set Speed</SoName>
  <SoLocation>room423</SoLocation>
  <Condition_Op></Condition_Op>
  <Condition_val>1</Condition_val>
  <zIndex>13</zIndex>
  <IsGroup>>false</IsGroup>
  <Parent ID>00000000-0000-0000-0000-000000000000</Parent ID>
</DesignerItem>

```

Figure 43: Fan Actuator XML Representation of BPMN

Although the XML file mentioned above is very important from the perspective of editing and updating the graphical models, it contains too much of unnecessary information from the perspective of BPM deployment and execution. For this purpose, every time a graphical BPM is stored by the user, another optimized version of the XML file is created with the sole purpose to be utilized by the BPM Deployment Engine. This XML files does not contain any information regarding the graphical rendering of the BPM and only provides information necessary for the deployment and execution of the process represented by the BPM. The XML sample is shown in Figure 39 representing tasks and other notations as DesignerItem objects. A DesignerItem tag in the figure completely represents the information encapsulated by a single BPM notation. In the figure first task represents a Task notation which encapsulates the complete information regarding a service object. The information include the names of the input, output devices associated with the SO, the complete URIs of the services for both the devices and the operational conditions for the execution of the SO. The file also includes connection objects to keep track of the source and sink items in the model and hence helps in identifying the correct sequence and execution order of the process.

## 4.4. IoT Smart Space Prototype

Table 7 show development Environment for IoT Smart Space Prototype.

Figure 44 presents finalized form of the smart space prototype. This prototype has been developed as a miniature representation of a smart space scenario where multiple sensing devices are deployed to capture the contextual data and an actuating device is deployed to modify the surroundings in the indoor environment. The prototype consists of the sensing and actuating devices shown in the figure which are used by users to customize the behavior of the smart space



based on the contextual situations. We utilize CoAP protocol to support the data and command transmission between the IoT proxy and IoT sensors. But for the communication between the IoT proxy and fan actuator module, we utilize the IoTivity protocol. The following illustrates the detail configuration of each IoT devices for the proposed smart space prototype.

**Table 7: Development Environment for IoT Smart Space Prototype**

<b>IoT Smart Space</b>	
Development Environment	Eclipse Luna SR2(4.4.2)
IoT Framework	OCF IoTivity
Machine Learning Software	Weka
Protocol	CoAP, IoTivity
Language	C, C++, Java
OS	Yocto Linux, Android
Hardware	i3-3220 CPU @3.30GHz, RAM 12.0 GB
IoT Platform	Intel Edison with Arduino* expansion board

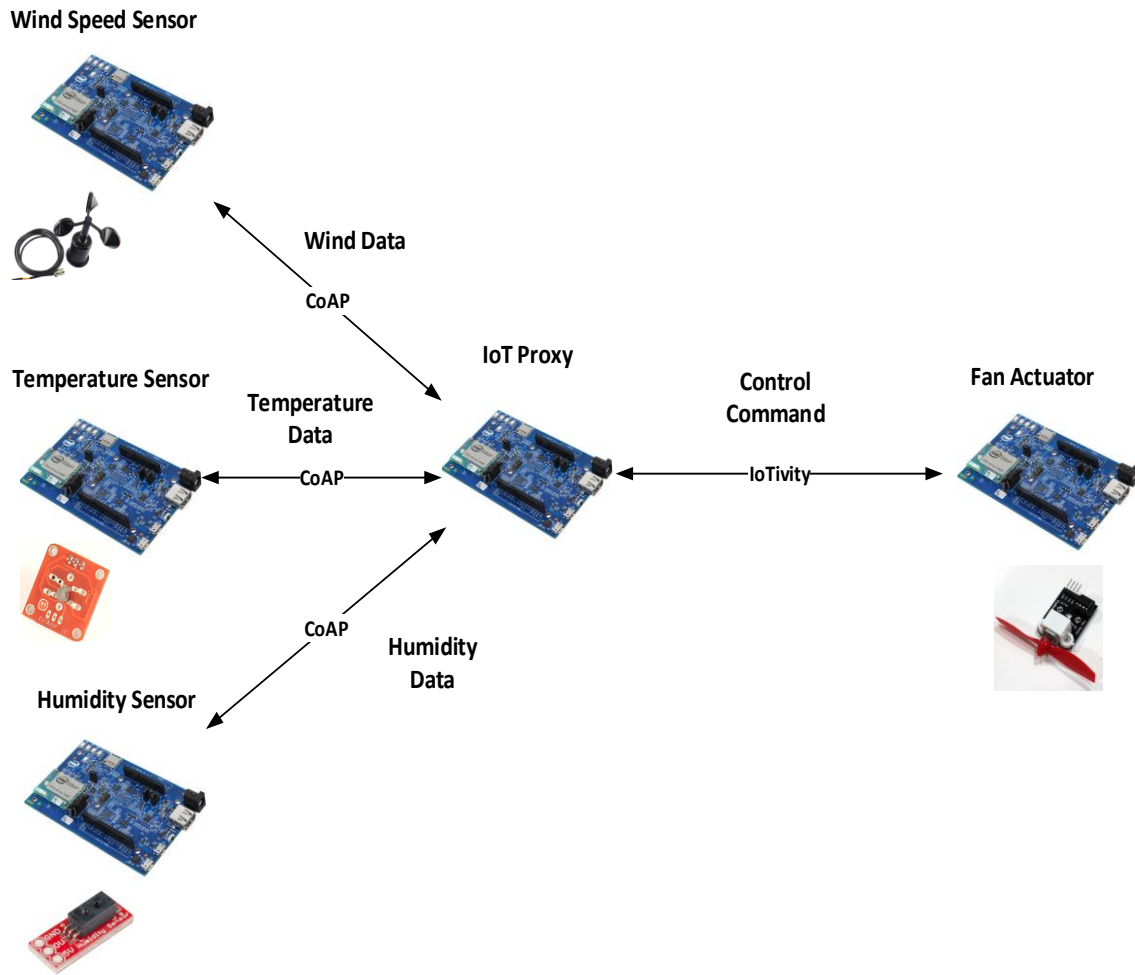
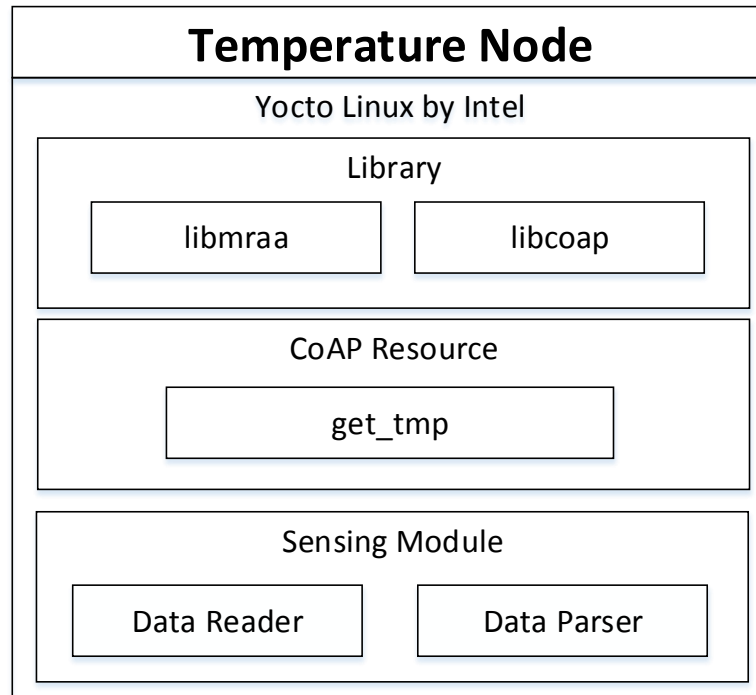


Figure 44: IoT Smart Space Application Prototype Structure

Figure 45 represents the configuration of temperature sensor with Intel Edison. The temperature sensor is implemented using Intel Edison with the Arduino breakout board. Sensing module is responsible for reading and parsing from raw sensing data. The temperature sensor has been used as CoAP resource (get\_tmp) as part of the CoAP server. For programming the CoAP server and defining the behavior of the CoAP resource, libcoap [56] library has been used. The library is based on C language so we had to use the Eclipse IoT Development Kit as the IDE for coding the CoAP device module. For interfacing the IO on Intel Edison, libmraa library is used.



**Figure 45: Implement Environment of Temperature Sensor**

Figure 46 represents the configuration of wind speed sensor with Intel Edison. The wind speed sensor is implemented using Intel Edison with the Arduino breakout board. Sensing module is responsible for reading and parsing from raw sensing data. The temperature sensor has been used as CoAP resource (`get_airflow`) as part of the CoAP server. For programming the CoAP server and defining the behavior of the CoAP resource, `libcoap` library has been used. The library is based on C language so we had to use the Eclipse IoT Development Kit as the IDE for coding the CoAP device module. For interfacing the IO on Intel Edison, `libmraa` library is used.

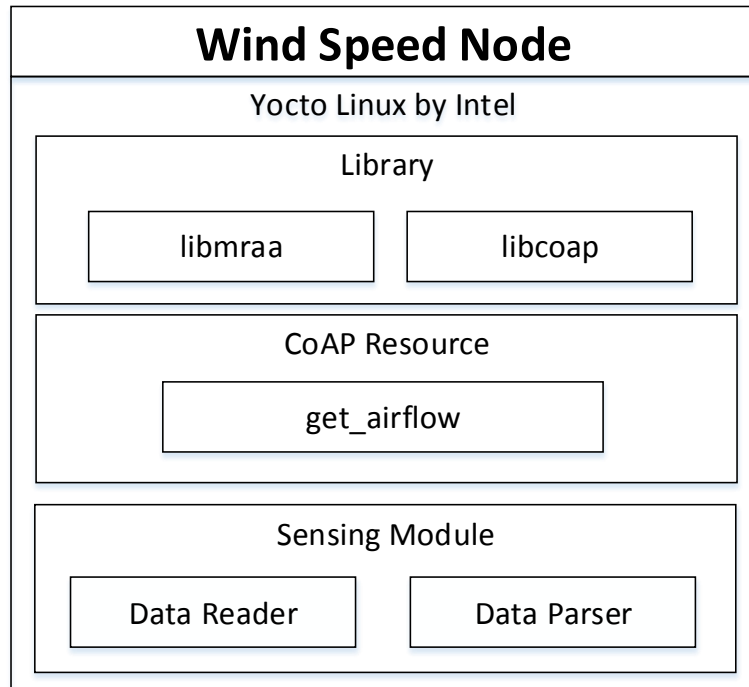


Figure 46: Implement Environment of Wind Speed Sensor

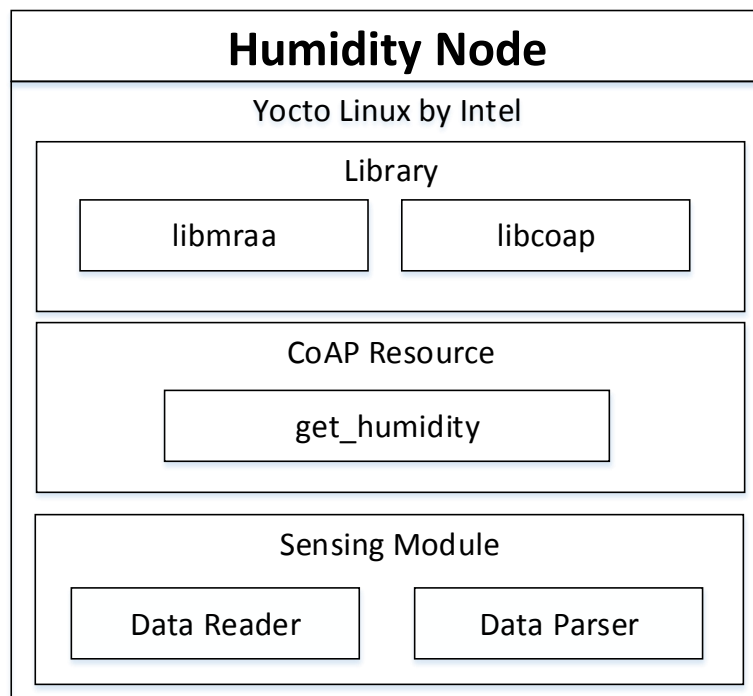
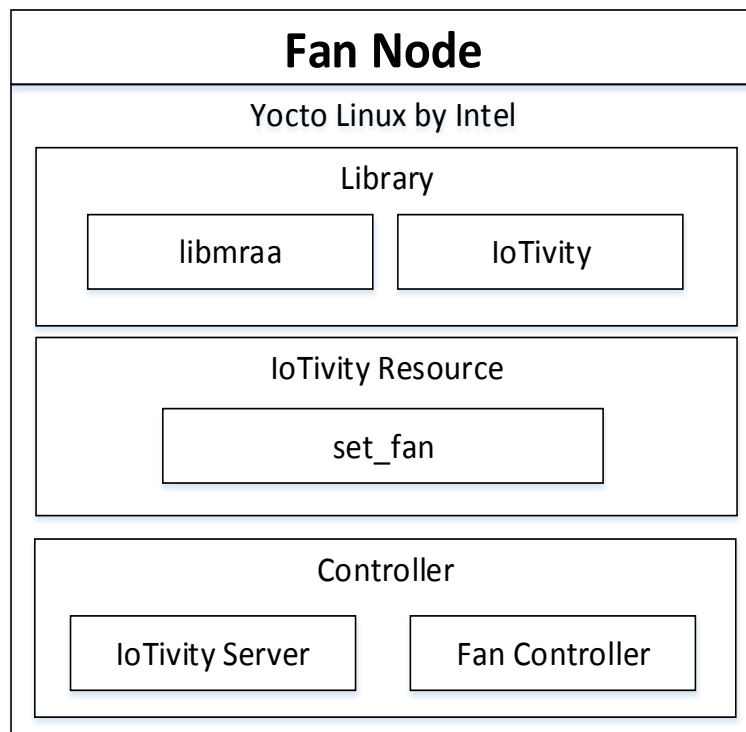


Figure 47: Implement Environment of Humidity Sensor

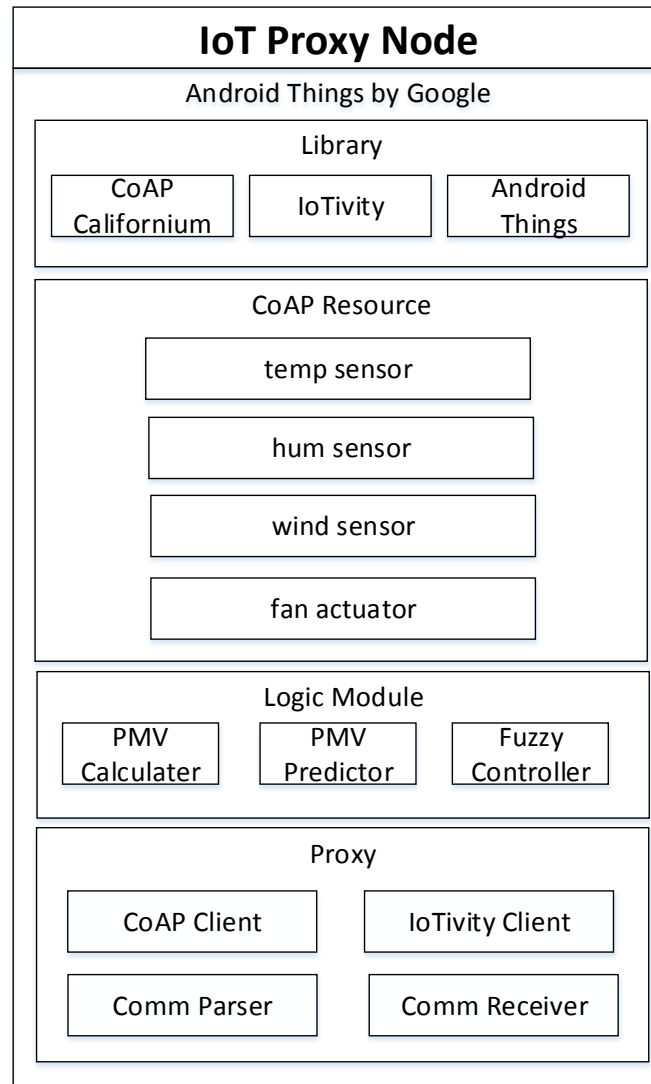
Figure 47 represents the configuration of humidity sensor with Intel Edison. The humidity sensor is implemented using Intel Edison with the Arduino breakout board. Sensing module is responsible for reading and parsing from raw sensing data. The temperature sensor has been used as CoAP resource (get\_humidity) as part of the CoAP server. For programming the CoAP server and defining the behavior of the CoAP resource, libcoap library has been used. The library is based on C language so we had to use the Eclipse IoT Development Kit as the IDE for coding the CoAP device module. For interfacing the IO on Intel Edison, libmraa library is used.



**Figure 48: Implement Environment of Fan Actuator**

Figure 48 represents the configuration of fan actuator with Intel Edison. The fan actuator is implemented using Intel Edison with the Arduino breakout board. Sensing module is responsible

for reading and parsing from raw sensing data. The temperature sensor has been used as IoTivity resource (set\_fan) as part of the IoTivity server. For programming the IoTivity server and defining the behavior of the IoTivity resource, IoTivity sdk has been used. The library is based on C++ language so we had to use the Eclipse IoT Development Kit as the IDE for coding the IoTivity device module. For interfacing the IO on Intel Edison, libmraa library is used.



**Figure 49: Implement Environment of IoT Proxy**

Figure 49 represents the configuration of IoT proxy with Intel Edison. The IoT proxy is implemented using Intel Edison with the Arduino breakout board. Android Things sdk has been

flashed into the Intel Edison to provide Android based environment. Logic module supports three services (PMV calculation, PMV prediction and fuzzy control) based on machine learning algorithms. Comm receiver is responsible for receiving the command from the virtual domain and the comm parser extracts the service process from the command. These IoT sensor and actuators have been used as CoAP resource as part of the CoAP server. For programming the CoAP server and defining the behavior of the CoAP resource, Californium library has been used. The library is based on Java language so that it can be directly imported in the Android environment. For programming the IoTivity server and defining the behavior of the IoTivity resource, IoTivity sdk has been used.

```

static double caIPMV(double ta, double tr, double vel, double rh, double met, double clo, double wme) {

    double pa, icl, m, w, mw, fcl, hcf, taa, tra, tc1a, p1, p2, p3, p4, p5, xn, xf, eps, hcn, hc = 1, tcl, h11, h12, h13, h14, h15, h16, ts, pmv, ppd, n;

    pa = rh * 10 * Math.exp(16.6536 - 4030.183 / (ta + 235));

    icl = 0.155 + clo; // thermal insulation of the clothing in M2K/W
    m = met + 58.15; // metabolic rate in W/M2
    w = wme + 58.15; // external work in W/M2
    mw = m - w; // internal heat production in the human body
    if (icl <= 0.078)
        fcl = 1 + (1.29 * icl);
    else
        fcl = 1.05 + (0.645 * icl);

    // heat transf. coeff. by forced convection
    hcf = 12.1 + Math.sqrt(vel);
    taa = ta + 273;
    tra = tr + 273;
    tc1a = taa + (35.5 - ta) / (3.5 * icl + 0.1);

    p1 = icl + fcl;
    p2 = p1 + 3.96;
    p3 = p1 + 100;
    p4 = p1 + taa;
    p5 = 308.7 - 0.028 * mw + p2 + Math.pow(tra / 100, 4);
    xn = tc1a / 100;
    xf = tc1a / 50;
    eps = 0.00015;
}

```

Figure 50: PMV Equation Parameter Definition

## 4.5. PMV Calculation

Figure 50 represents the source code of the PMV equation parameter definition in Java. The air temperature, mean radiant temperature, relative air velocity, relative humidity, metabolic rate, clothing and external work parameter are defined as  $t_a$ ,  $t_r$ ,  $vel$ ,  $rh$ ,  $met$ ,  $clo$  and  $wme$  in the beginning of the function. Then the thermal insulation value of the clothing is calculated according to the metabolic rate and external work values. Clothing factor value varies on the basis of clothing insulation. Heat transfer coefficient is calculated by forced convection.

```
tcl = 100 + xn - 273;

// heat loss diff. through skin
h11 = 3.05 + 0.001 * (5733 - (6.99 + mw) - pa);
// heat loss by sweating
if (mw > 58.15)
|   h12 = 0.42 * (mw - 58.15);
else
|   h12 = 0;
// latent respiration heat loss
h13 = 1.7 + 0.00001 * m * (5867 - pa);
// dry respiration heat loss
h14 = 0.0014 * m * (34 - ta);
// heat loss by radiation
h15 = 3.96 * fcl * (Math.pow(xn, 4) - Math.pow(tra / 100, 4));
// heat loss by convection
h16 = fcl + hc * (tcl - ta);

ts = 0.303 + Math.exp(-0.036 * m) + 0.028;
pmv = ts * (mw - h11 - h12 - h13 - h14 - h15 - h16);
ppd = 100.0 - 95.0 + Math.exp(-0.03353 + Math.pow(pmv, 4.0)) - 0.2179
|   + Math.pow(pmv, 2.0));

return pmv;
```

Figure 51: PMV Equation Implementation

Figure 51 illustrates the source code of the PMV equation implementation in Java. All parameters used in the PMV equation are calculated according to the following figure. Heat



loss difference value is calculated by air pressure and human body internal heat production. Heat loss by sweating varies on the basis of human body internal heat production. Then latent respiration, dry respiration, radiation heat loss and convection heat loss are calculated in sequence. These parameters are used in the PMV equation to get the PMV index value.

Figure 52 represents the execution result of PMV index calculation. T, vel and rh represent three parameters (temperature, wind speed and humidity). And then the IoT Proxy invokes the function in the previous figure to get the PMV index value.

```
I/System.out: t: 18.58  
I/System.out: vel: 2.0  
I/System.out: rh: 30.58  
I/System.out: PMV: -1.8
```

**Figure 52: PMV Execution Result**

## 4.6. PMV Prediction based on Linear Regression

Figure 53 represents the training dataset for PMV index prediction in the proposed IoT prototype. An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. The first section of the ARFF file is the Header information which contains the name of the relation (indoor\_pmv), a list of the attributes (the columns in the data), and their types. The @RELATION, @ATTRIBUTE and @DATA declarations are case insensitive. The relation name is defined as the first line in the ARFF file. The format is @relation <relation-name> where <relation-name> is a string. Attribute declarations take the form of an ordered sequence of @attribute statements. Each attribute in the data set has its own @attribute statement which uniquely defines the name of that attribute and its data type. The order the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all that attributes values will be found

in the third comma delimited column. The format for the @attribute statement is: @attribute <attribute-name> <datatype> where the <attribute-name> must adhere to the constraints specified in the above section on the @relation declaration. The <datatype> can be any of the four types supported by Weka:

- numeric
- integer is treated as numeric
- real is treated as numeric
- <nominal-specification>
- string
- date [<date-format>]
- relational for multi-instance data (for future use)

where <nominal-specification> and <date-format> are defined below. The keywords numeric, real, integer, string and date are case insensitive. The @data declaration is a single line denoting the start of the data segment in the file. The format is: @data. Each instance is represented on a single line, with carriage returns denoting the end of the instance. A percent sign (%) introduces a comment, which continues to the end of the line. Attribute values for each instance can be delimited by commas or tabs. A comma/tab may be followed by zero or more spaces. Attribute values must appear in the order in which they were declared in the header section (i.e., the data corresponding to the nth @attribute declaration is always the nth field of the attribute). The training dataset used in this prototype has one hundred lines, containing three attribute features (temperature, humidity and wind speed values) and one class label (pmv).

```

1 @relation indoor_pmv
2
3 @attribute temperature numeric
4 @attribute humidity numeric
5 @attribute 'wind speed' numeric
6 @attribute pmv numeric
7
8 @data
9 24.66,48.86,0,0.39
10 28.71,40.94,0,0.88
11 18.14,4.5,0,-0.36
12 25.62,49.24,0,0.51
13 10.91,26.75,0,-1.16
14 17.42,53.09,0,-0.44
15 25.8,50.9,1,0.53
16 19.03,0.92,1,-0.26
17 15.32,40.77,1,-0.68
18 28.07,43.5,1,0.8
19 25.54,34,0,0.43
20 25.95,42.7,1,0.55
21 19,31.79,0,-0.27
22 13.75,16.79,0,-0.85
23 19.71,4.09,0,-0.18
24 17.82,36.63,0,-0.4
25 19.21,5.02,1,-0.24
26 25.23,54.06,1,0.46
27 27.09,45.38,0,0.68
28 16.14,46.62,0,-0.59
29 21.18,12.42,1,-0.02
30 29.83,8.16,0,1.02
31 22.85,23.33,1,0.18
32 17.82,20.73,0,-0.4
33 21.89,10.94,1,0.07
34 24.32,14.99,0,0.35
35 26.59,59.6,0,0.62
36 29.17,28.06,0,0.94
37 22.2,43.23,0,0.1
38 20.71,58.88,0,-0.07
39 25.96,49.53,0,0.55
40 26.15,36.46,1,0.57
41 27.5,46.05,0,0.73
42 10.8,0.48,0,-1.18
43 22.98,36.73,1,0.19
44 11.97,54.66,1,-1.05
45 24.71,24.22,0,0.4
46 11.24,46.51,1,-1.13
47 10.95,33.61,1,-1.16
48 20.97,27.7,1,-0.04
49 20.62,0.69,1,-0.08
50 26.14,37.5,0,0.57
51 11.77,46.64,0,-1.07
52 10.94,41.11,0,-1.16
53 12.55,57.61,1,-0.98
54 29.94,34.34,0,1.03
55 24.02,28.87,0,0.32
56 10.28,55.44,1,-1.23
57 23.74,19.53,0,0.28
58 24.09,36.11,1,0.32
59 14.72,26.5,0,-0.75

```

Figure 53: Snapshot of Training Dataset for PMV Index Prediction

Figure 54 presents the source code of linear regression based PMV index prediction process using Weka. To load data into Weka, we have to put it into a format that will be understood. Weka's preferred method for loading data is in the ARFF. The ARFF file that we used is represented in Figure 48. After loading the training dataset, we can simply invoke the linear regression class which is encapsulated by Weka. We build the linear regression module with the training dataset and then go to the next step. To predict the PMV index value, we have to load the test dataset which contains the current environmental data. Weka supports classify instance function which returns the numeric value of the class to predict.

```

private void LrPrediction() throws Exception {
    System.out.println("Start Prediction");
    // load dataset
    ConverterUtils.DataSource source = new ConverterUtils.DataSource(MainActivity.getAssets().open("indoor_pmv.arff"));
    Instances Dataset = source.getDataSet();
    System.out.println(Dataset.toSummaryString());
    Dataset.setClassIndex(Dataset.numAttributes()-1);

    //build model
    LinearRegression lr = new LinearRegression();
    lr.buildClassifier(Dataset);
    // output model
    System.out.println(lr);

    //load new dataset
    System.out.println("Load Test Dataset");
    File file = new File(MainActivity.getFilesDir(), "pmv_test.arff");
    FileInputStream fis = new FileInputStream(file);
    ConverterUtils.DataSource testSource = new ConverterUtils.DataSource(fis);
    Instances testDataset = testSource.getDataSet();
    System.out.println(testDataset.toSummaryString());
    testDataset.setClassIndex(testDataset.numAttributes()-1);
    // loop through the new dataset and make predictions
    System.out.println("-----");
    System.out.println("Actual Class, LR Predicted");
    for (int i = 0; i < testDataset.numInstances(); i++) {
        Instance newInst = testDataset.instance(i);
        // call classifyInstance, which returns a double value for the class
        predLR = lr.classifyInstance(newInst);
        System.out.println("predicted value: "+ predLR);
    }
}

```

Figure 54: Snapshot of Source Code for PMV Index Prediction Process using Weka

Figure 55 illustrates the prediction execution result of PMV index in the IoT Proxy. We use current indoor environment conditions to build test dataset that is used to predict the PMV index value.

```

I/System.out: Actual Class, LR Predicted
I/System.out: predicted value: -0.30045871257347523

```

Figure 55: PMV Index Prediction Execution Result

## 4.7. Control based on Fuzzy Logic with PMV

To implement the fuzzy inference system in the IoT prototype, we utilized the jFuzzyLogic library [57] implementing Fuzzy Control language (FCL) which standardizes programming fuzzy logic systems. In our fuzzy inference system, the input variables are pmv index and its variation

(deltapmv). Variables are defined in the VAR\_INPUT and VAR\_OUTPUT sections. Fuzzy sets are defined in FUZZIFY blocks for input variables and DEFUZZIFY blocks for output variables. One FUZZIFY block is used for each input variable. Each TERM line within a FUZZIFY block defines a linguistic term and its corresponding membership function. Output variables define their membership functions within DEFUZZIFY blocks. Linguistic terms and membership functions are defined using the TERM keyword as previously described for input variables. The statement 'METHOD : COG' indicates that we are using 'Center of gravity'.

```

FUNCTION_BLOCK tipper // Block definition (there may be more than one block per file)

VAR_INPUT // Define input variables
    pmv : REAL;
    deltapmv : REAL;
END_VAR

VAR_OUTPUT // Define output variable
    fanspeed : REAL;
END_VAR

FUZZIFY pmv
    TERM tooCold := (-3, 0) (-0.9, 1)(-0.4,0);
    TERM winterComfort := (-0.6, 0) (-0.3,1) (0,0);
    TERM neutral := (-0.2, 0) (0, 1)(0.2,0);
    TERM summerComfort := (0,0)(0.4,1)(0.7,0);
    TERM tooHot := (0.4,0)(1,1)(3,0);
END_FUZZIFY

FUZZIFY deltapmv
    TERM negativeBig := (-3, 0) (-0.5, 1) (-0.3,0) ;
    TERM negativeMedium := (-0.5,0) (-0.3,1)(-0.1,0);
    TERM negativeSmall := (-0.2,0) (-0.1,1)(0,0);
    TERM zero := (-0.05,0) (0.01,1)(0.05,0);
    TERM positiveSmall := (0,0) (0.1,1)(0.2,0);
    TERM positiveMedium := (0.1,0) (0.3,1)(0.5,0);
    TERM positiveBig := (0.3,0) (0.5,1)(3,0);
END_FUZZIFY

DEFUZZIFY fanspeed
    TERM zero := (0,0) (0.1,1) (0.2,0);
    TERM low := (0.1,0) (0.2,1) (0.4,0);
    TERM medium := (0.2,0) (0.4,1) (0.7,0);
    TERM high := (0.4,0) (0.6,1) (0.9,0);
    TERM veryHigh := (0.8,0) (1,1) (1,0);
    METHOD : COG; // Use 'Center Of Gravity' defuzzification method
    DEFAULT := 0; // Default value is 0 (if no rule activates defuzzifier)
END_DEFUZZIFY

```

Figure 56: Fuzzy Controller Variable Definitions

Fuzzy rules are defined in RULEBLOCK statements. Each entry in the RB was converted to a single FCL rule. Within each rule, the antecedent (i.e. the IF part) is composed of the input variables connected by ‘AND’ operators. Since there is more than one output variable, we can specify multiple consequents (i.e. THEN part) separated by semicolons.

```

RULEBLOCK No1
  AND : MIN;           // Use 'min' for 'and' (also implicit use 'max' for 'or' to fulfill DeMorgan's Law)
  ACT : MIN;           // Use 'min' activation method
  ACCU : MAX;          // Use 'max' accumulation method

  RULE 1 : IF pmv IS tooHot AND delpmv IS negativeSmall THEN fanspeed IS veryHigh;
  RULE 2 : IF pmv IS tooHot AND delpmv IS negativeMedium THEN fanspeed IS veryHigh;
  RULE 3 : IF pmv IS tooHot AND delpmv IS negativeBig THEN fanspeed IS high;
  RULE 4 : IF pmv IS tooHot AND delpmv IS zero THEN fanspeed IS high;
  RULE 5 : IF pmv IS tooHot AND delpmv IS positiveSmall THEN fanspeed IS veryHigh;
  RULE 6 : IF pmv IS tooHot AND delpmv IS positiveMedium THEN fanspeed IS veryHigh;
  RULE 7 : IF pmv IS tooHot AND delpmv IS positiveBig THEN fanspeed IS high;
  RULE 8 : IF pmv IS summerComfort AND delpmv IS negativeSmall THEN fanspeed IS veryHigh;
  RULE 9 : IF pmv IS summerComfort AND delpmv IS negativeMedium THEN fanspeed IS high;
  RULE 10 : IF pmv IS summerComfort AND delpmv IS negativeBig THEN fanspeed IS high;
  RULE 11 : IF pmv IS summerComfort AND delpmv IS zero THEN fanspeed IS high;
  RULE 12 : IF pmv IS summerComfort AND delpmv IS positiveSmall THEN fanspeed IS veryHigh;
  RULE 13 : IF pmv IS summerComfort AND delpmv IS positiveMedium THEN fanspeed IS high;
  RULE 14 : IF pmv IS summerComfort AND delpmv IS positiveBig THEN fanspeed IS high;
  RULE 15 : IF pmv IS neutral AND delpmv IS negativeSmall THEN fanspeed IS medium;
  RULE 16 : IF pmv IS neutral AND delpmv IS negativeMedium THEN fanspeed IS low;
  RULE 17 : IF pmv IS neutral AND delpmv IS negativeBig THEN fanspeed IS low;
  RULE 18 : IF pmv IS neutral AND delpmv IS zero THEN fanspeed IS low;
  RULE 19 : IF pmv IS neutral AND delpmv IS positiveSmall THEN fanspeed IS medium;
  RULE 20 : IF pmv IS neutral AND delpmv IS positiveMedium THEN fanspeed IS low;
  RULE 21 : IF pmv IS neutral AND delpmv IS positiveBig THEN fanspeed IS low;
  RULE 22 : IF pmv IS winterComfort AND delpmv IS negativeSmall THEN fanspeed IS low;
  RULE 23 : IF pmv IS winterComfort AND delpmv IS negativeMedium THEN fanspeed IS zero;
  RULE 24 : IF pmv IS winterComfort AND delpmv IS negativeBig THEN fanspeed IS zero;
  RULE 25 : IF pmv IS winterComfort AND delpmv IS zero THEN fanspeed IS zero;
  RULE 26 : IF pmv IS winterComfort AND delpmv IS positiveSmall THEN fanspeed IS low;
  RULE 27 : IF pmv IS winterComfort AND delpmv IS positiveMedium THEN fanspeed IS zero;
  RULE 28 : IF pmv IS winterComfort AND delpmv IS positiveBig THEN fanspeed IS zero;
  RULE 29 : IF pmv IS tooCold AND delpmv IS negativeSmall THEN fanspeed IS low;
  RULE 30 : IF pmv IS tooCold AND delpmv IS negativeMedium THEN fanspeed IS zero;
  RULE 31 : IF pmv IS tooCold AND delpmv IS negativeBig THEN fanspeed IS zero;
  RULE 32 : IF pmv IS tooCold AND delpmv IS zero THEN fanspeed IS zero;
  RULE 33 : IF pmv IS tooCold AND delpmv IS positiveSmall THEN fanspeed IS low;
  RULE 34 : IF pmv IS tooCold AND delpmv IS positiveMedium THEN fanspeed IS zero;
  RULE 35 : IF pmv IS tooCold AND delpmv IS positiveBig THEN fanspeed IS zero;
END_RULEBLOCK

```

Figure 57: Fuzzy Controller Fuzzy Rule Block

Figure 58 presents the execution result of the fuzzy controller. PMV index and PMV variation values are used as input parameters in the fuzzy controller. And then according to the defined fuzzy rules, the fuzzy controller computes the output value (fan speed).

```
I/System.out: Fan Speed 0.1796373218192193  
I/System.out: duty:83
```

**Figure 58: Fuzzy Controller Execution Result**

## 5. Experiment and Evaluation

Figure 59 represents the experiment scenario of the proposed smart indoor space prototype. For the purpose of the smart space prototype demonstration, the temperature sensor, humidity sensor, wind sensor, proxy and the fan actuator are implemented as IoT devices. The physical layer of the architecture represents these components as separate physical devices. The information of the implemented device components is provided to the Virtual Object Layer which converts the information into virtual objects. The Service Logic Layer utilizes the virtual objects generated from the Virtual Object Layer and the logic objects from the service object repository to combine logic service objects. The Business Process Layer acquires the service object definitions from the SO repository at Service Layer, parses the XML representation of the service objects to extract information and then represent the services as BPMN task notations. The created BPM is then deployed to the proxy and control the IoT devices on the basis of the operations.

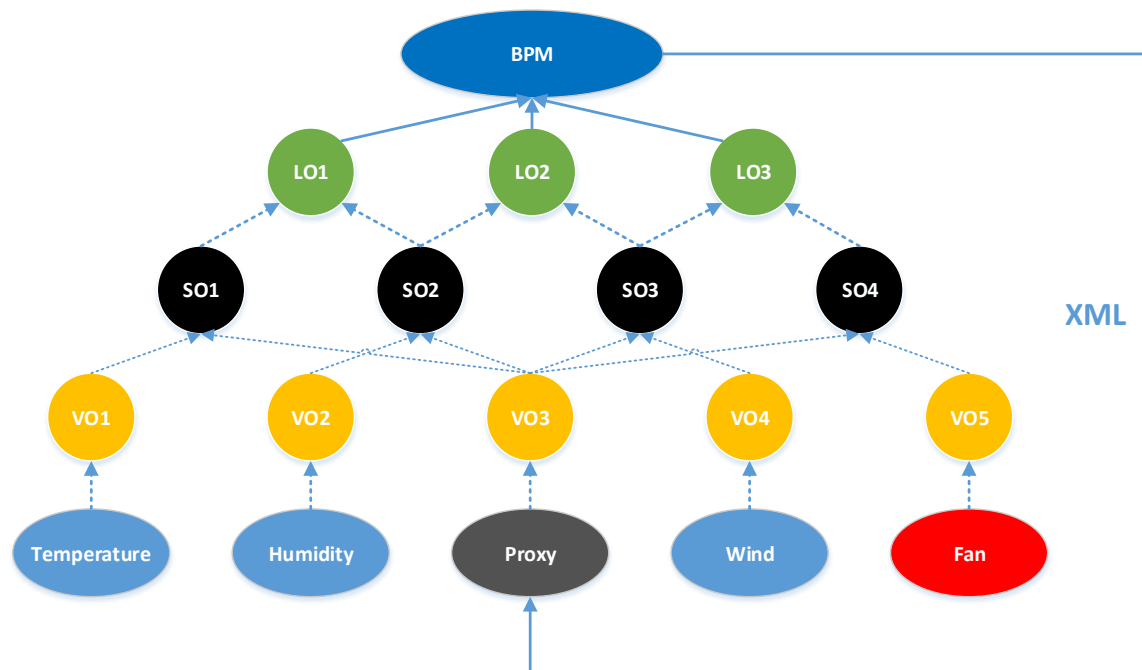
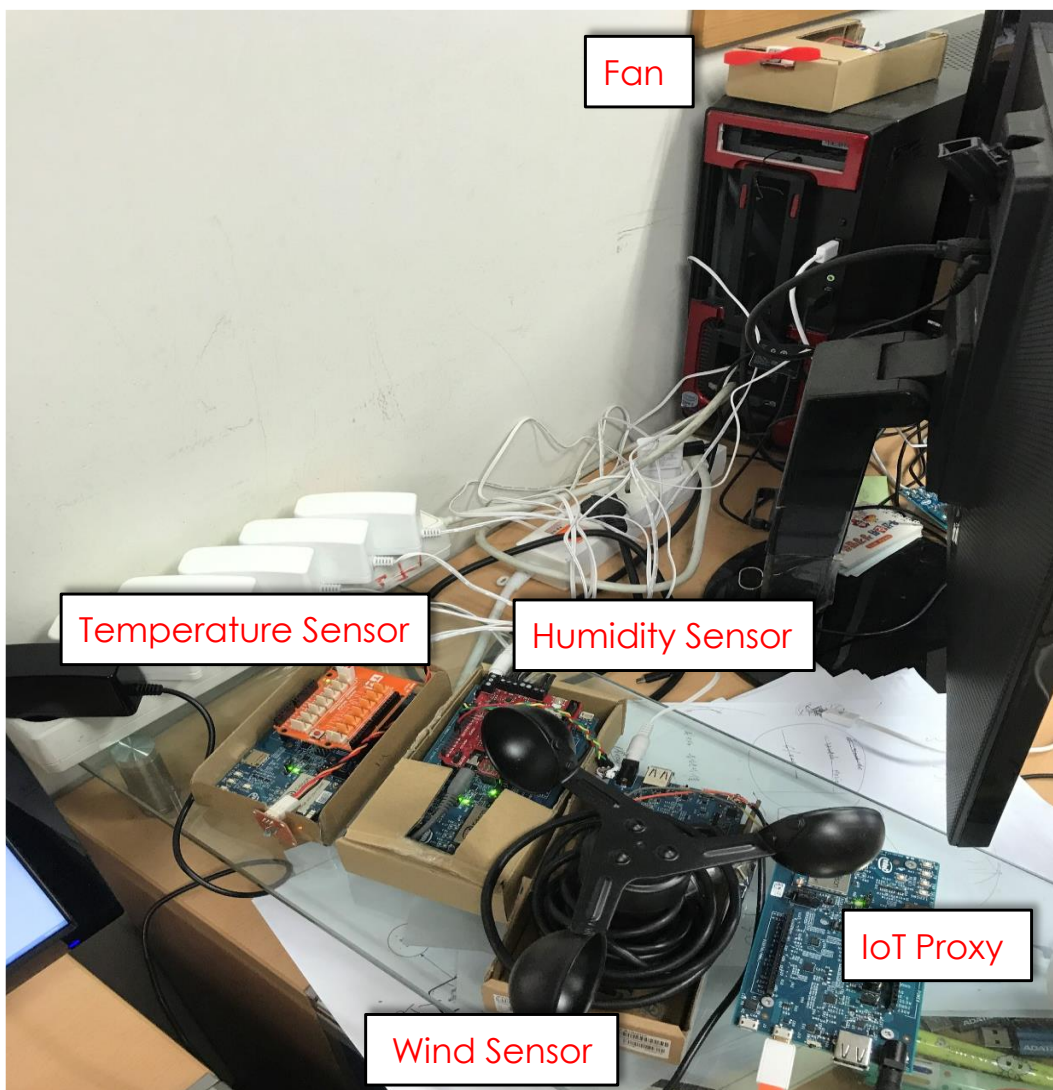


Figure 59: Experiment Scenario of the Smart Indoor Space Prototype



Figure 60 presents finalized form of the smart space prototype. This prototype has been developed as a miniature representation of a smart space scenario where multiple sensing devices are deployed to capture the contextual data and an actuating device is deployed to modify the surroundings in the indoor environment. The prototype consists of the following sensing and actuating devices which are used by users to customize the behavior of the smart space based on the contextual situations.



**Figure 60: Intel Edison based Finalized Smart Indoor Space Prototype for Experiment**

Figure 61 represents the execution result of BPM process in the IoT Proxy. The BPM file shown in the previous section that is sent from the BPM Editor. These xml documents can be received and executed by the IoT Proxy. The file is parsed to extract all the executable entities as represented by the BPM notations. Based on the predefined connection between the notations, the executable entities are sorted and sequenced so that the final execution of the process is in synch with the original graphical BPM process created by the user. The task entities are sorted and each entity is executed. For task related to remote IoT resources, the IoT Proxy then requests the CoAP resource of corresponding the IoT devices. Once the CoAP devices receives the request from the IoT proxy, the response based on the complete execution is then sent back to the IoT Proxy.

```

I/System.out: *****
I/System.out: Receiving BPM XML File
I/System.out: <Root><DesignerItem><Left>0</Left><ID>c5f20854-26d1-4323-b837-b9c1e3832321</ID><ItemType>START
I/System.out: *****
I/System.out: *****
I/System.out: Start Parsing BPM XML File
I/System.out: *****
I/System.out: 1: dev type = Temperature uri = coap://192.168.0.21:5683/temperature/ service = GetTempC
I/System.out: 2: dev type = Humidity uri = coap://192.168.0.15:5683/humidity/ service = GetHum
I/System.out: 3: dev type = Wind uri = coap://192.168.0.11:5683/GetWind/ service = GetWind
I/System.out: 4: dev type = PMV uri = coap://coap://192.168.0.5:5683/pmv/ service = GetPMV
I/System.out: 5: dev type = LROutput uri = coap://coap://192.168.0.5:5683/LrPred/ service = StartPrediction
I/System.out: 6: dev type = FuzzyOutput uri = coap://coap://192.168.0.5:5683/FisControl/ service = StartFis
I/System.out: 7: dev type = Fan uri = coap://192.168.0.10:5683/SetFan/ service = SetSpeed

```

**Figure 61: BPM Execution Result in IoT Proxy**

This figure presents the PMV index prediction process in the IoT proxy. After getting sensing values from IoT sensors, the IoT proxy calls the function to store sensing values in the ARFF format that is used for testing. Then the IoT proxy loads the pre-defined training dataset and build linear regression model according to the features information.

```

I/System.out: coap://192.168.0.14:5683/temperature
I/System.out: ==[ CoAP Request ]=====
I/System.out: MID      : 26572
I/System.out: Token    : 1ef452d0
I/System.out: Type     : CON
I/System.out: Method   : GET
I/System.out: Options: {"Uri-Path":"pmv"}
I/System.out: Payload: 0 Bytes
I/System.out: =====
I/System.out: temperature: 18.58
I/System.out: coap://192.168.0.15:5683/humidity
I/System.out: ==[ CoAP Request ]=====
I/System.out: MID      : 26572
I/System.out: Token    : 1ef452d0
I/System.out: Type     : CON
I/System.out: Method   : GET
I/System.out: Options: {"Uri-Path":"pmv"}
I/System.out: Payload: 0 Bytes
I/System.out: =====
I/System.out: humidity: 30.58
I/System.out: coap://192.168.0.11:5683/windspeed
I/System.out: ==[ CoAP Request ]=====
I/System.out: MID      : 26572
I/System.out: Token    : 1ef452d0
I/System.out: Type     : CON
I/System.out: Method   : GET
I/System.out: Options: {"Uri-Path":"pmv"}
I/System.out: Payload: 0 Bytes
I/System.out: =====
I/System.out: wind speed: 2

```

**Figure 62: Sensing Response Information in IoT Proxy**

```

I/System.out: t: 18.58
I/System.out: vel: 2.0
I/System.out: rh: 30.58
I/System.out: PMV: -1.8
I/System.out: @relation pmv_test-weka.filters.unsupervised.instance.NonSparseToSparse
I/System.out: @attribute temperature numeric
I/System.out: @attribute humidity numeric
I/System.out: @attribute 'wind speed' numeric
I/System.out: @attribute pmv numeric
I/System.out: @data
I/System.out: {0 18.58,1 30.58,2 2}
I/System.out: /data/user/0/com.example.androidthings.myproject/files
I/System.out: Start Prediction
I/System.out: Relation Name: indoor_pmv
I/System.out: Num Instances: 100
I/System.out: Num Attributes: 4
I/System.out:


| Name          | Type | Nom | Int  | Real | Missing | Unique   | Dist |
|---------------|------|-----|------|------|---------|----------|------|
| 1 temperature | Num  | 0%  | 2%   | 98%  | 0 / 0%  | 96 / 96% | 98   |
| 2 humidity    | Num  | 0%  | 1%   | 99%  | 0 / 0%  | 96 / 96% | 98   |
| 3 wind speed  | Num  | 0%  | 100% | 0%   | 0 / 0%  | 0 / 0%   | 2    |
| 4 pmv         | Num  | 0%  | 1%   | 99%  | 0 / 0%  | 73 / 73% | 85   |


I/System.out: Linear Regression Model
I/System.out: pmv =
I/System.out: 0.1148 * temperature +
I/System.out: -2.4333

```

**Figure 63: Built Linear Regression Model for PMV Index Prediction in IoT Proxy**

The predicted PMV index value and the PMV variation values are used as input parameters in Fuzzy Inference System. Then the IoT proxy initializes the Fuzzy Inference System and the fuzzy sets are shown in the following figure.

```

I/System.out: Actual Class, LR Predicted
I/System.out: predicted value: -0.30045871257347523
I/System.out: Start Fuzzy Inference System
I/System.out: =====
I/System.out: 0.47446181836863444
I/System.out: FUNCTION_BLOCK tipper
I/System.out: VAR_INPUT
I/System.out:   deltapmv : REAL;
I/System.out:   pmv : REAL;
I/System.out: END_VAR
I/System.out: VAR_OUTPUT
I/System.out:   fanspeed : REAL;
I/System.out: END_VAR
I/System.out: FUZZIFY deltapmv
I/System.out:   TERM negativeBig := (-3.0, 0.0) (-0.5, 1.0) (-0.3, 0.0) ;
I/System.out:   TERM negativeMedium := (-0.5, 0.0) (-0.3, 1.0) (-0.1, 0.0) ;
I/System.out:   TERM negativeSmall := (-0.2, 0.0) (-0.1, 1.0) (0.0, 0.0) ;
I/System.out:   TERM positiveBig := (0.3, 0.0) (0.5, 1.0) (3.0, 0.0) ;
I/System.out:   TERM positiveMedium := (0.1, 0.0) (0.3, 1.0) (0.5, 0.0) ;
I/System.out:   TERM positiveSmall := (0.0, 0.0) (0.1, 1.0) (0.2, 0.0) ;
I/System.out:   TERM zero := (-0.05, 0.0) (0.01, 1.0) (0.05, 0.0) ;
I/System.out: END_FUZZIFY
I/System.out: FUZZIFY pmv
I/System.out:   TERM neutral := (-0.2, 0.0) (0.0, 1.0) (0.2, 0.0) ;
I/System.out:   TERM summerComfort := (0.0, 0.0) (0.4, 1.0) (0.7, 0.0) ;
I/System.out:   TERM tooCold := (-3.0, 0.0) (-0.9, 1.0) (-0.4, 0.0) ;
I/System.out:   TERM tooHot := (0.4, 0.0) (1.0, 1.0) (3.0, 0.0) ;
I/System.out:   TERM winterComfort := (-0.6, 0.0) (-0.3, 1.0) (0.0, 0.0) ;

```

**Figure 64: Launch Fuzzy Inference System in IoT Proxy**

Fan speed value is the output of the Fuzzy Inference System. And then the fan speed value is then converted to duty which is used as a query value in the control request.

```

I/System.out: Fan Speed 0.10000000000000002
I/System.out: duty:90

```

**Figure 65: Fuzzy Inference System Output in IoT Proxy**

The proposed IoT smart indoor space prototype has been tested in a working space (office) in the south of Korea. Table 8 records the fan speed variation at 2:00 pm and it continues until 1:00

am through the day in the summer situation. Table depicts the experimental test performed to verify if the fan actuator operates correctly according to the environment situation. Fan speed values vary on basis of the predicted PMV index and PMV variation values. The fan speed increases with the temperature increasing and decreases as the temperature decreasing. According to the table, the air temperature values are the most important factor to affect the PMV index. The fuzzy inference system computes the speed to drive the fan coil according to the defined fuzzy rules shown in Figure 53.

**Table 8: Smart indoor space prototype performance**

Time	Temperature (°C)	Humidity (%)	Wind Speed (m/s)	PMV	Predicted PMV	PMV Variation	Fan Speed
2:00 pm	28.71	40.94	0	0.8	0.86	0	0.93
3:00 pm	22.14	37.5	0	-0.09	0.11	-0.75	0.49
4:00 pm	25.62	40.24	0	0.61	0.64	0.53	0.64
5:00 pm	20.91	36.75	0	-0.02	-0.03	-0.67	0.22
6:00 pm	25.8	40.9	1	0.53	0.52	0.55	0.65
7:00 pm	23.03	38.92	1	0.19	0.21	-0.31	0.64
8:00 pm	22.32	40.77	1	0.12	0.13	-0.08	0.60
9:00 pm	27.07	43.5	1	0.66	0.67	0.54	0.65
10:00 pm	25	44.34	0	0.42	0.44	-0.37	0.64
11:00 pm	22.75	36.79	0	0.17	0.18	-0.26	0.58
12:00 pm	20.94	34.34	0	-0.02	-0.03	-0.21	0.22
1:00 am	18.72	36.5	0	-0.26	-0.28	-0.25	0.07

As shown in the Figure 66, we measured the whole prototype system round trip time by the following equation:

$$R_t = \frac{1}{n} \sum_{i=0}^n t_n \quad (13)$$

Where  $R_t$  the round trip time and  $\{t=1, \dots, n\}$  is the time slots between the temperature sensor and the proxy, the humidity sensor and the proxy, the wind sensor and the proxy, the processing

time in the proxy, the proxy and the fan actuator. According to the Figure 67, we have recorded the round trip time of the whole prototype five times. And the average round trip time is 364.4ms that indicates CoAP and IoTivity protocol makes high performance in the transmission between IoT devices.

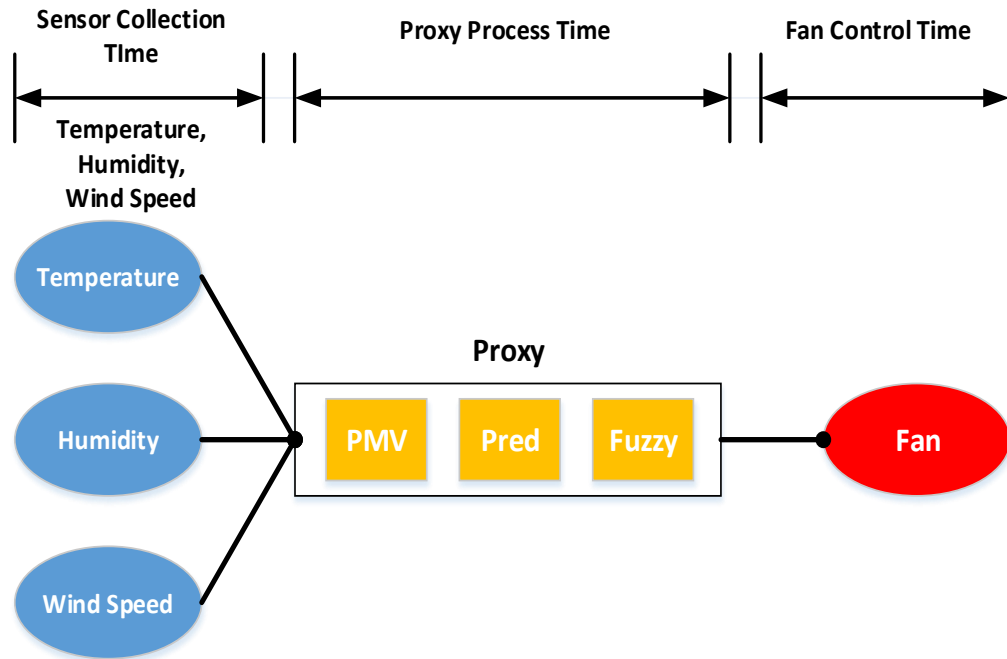
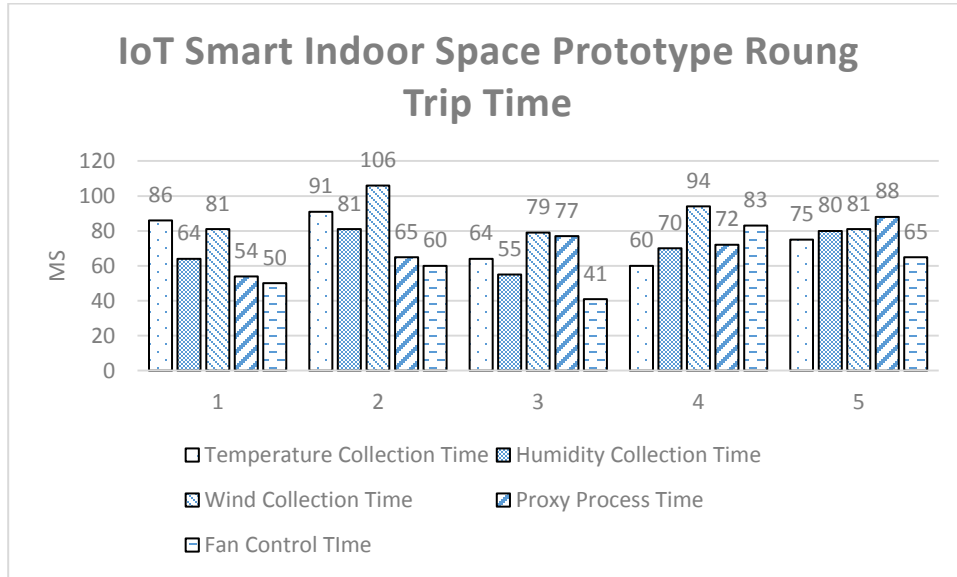


Figure 66: Smart Indoor Space Prototype Evaluation Structure



**Figure 67: IoT Smart Indoor Space Prototype Round Trip Time**



## 6. Conclusion

This paper presented an enhanced IoT cooperation architecture based on Business Process Modeling approach. The presented system utilizes the prevalent ideas of object virtualization and service provisioning and represents them as a DIY interface based on a Business Process Modeling (BPM). BPM has been at the core of software requirement analysis and specification processes. It fills the communication gap between the clients and developers by providing a standardized set of graphical notation called as the Business Process Modeling Notations (BPMN). BPMN is easy to learn and can be globally interpreted into the same description of a process. Although there are efforts to extend BPMN to incorporate IoT concepts but we believe that a standardized modeling language such as BPMN can provide better DIY environment for IoT application development. It is specifically important from the IoT point of view because IoT is an emerging field and mass involvement has been reported to be very necessary for the successful realization of IoT vision.

The paper visualizes the presented idea in the form of a layered architecture which consists of the Physical Cooperation Network Layer, Virtual Object Layer, Service Logic Layer, Business Process Layer and Application Layer. A detailed description of the layered architecture and design detail of the layers has been presented in this document. In order to demonstrate the applicability and usability of the proposed architecture in general IoT scenarios, an IoT smart space prototype has been implemented through the proposed architecture. Prototype implementations based on multiple CoAP devices and intelligent services (prediction and automatic control) have been developed to demonstrate the feasibility of the proposed system.

## References

1. Internet of things, [https://en.wikipedia.org/wiki/Internet\\_of\\_things](https://en.wikipedia.org/wiki/Internet_of_things).
2. "IoT Applications With Examples", <http://internetofthingswiki.com/iot-applications-examples/541/>.
3. Gartner, "Predicts 2015: The Internet of Things", December 2014.
4. Mateo P. Internet of Things Needs Standards in Order to Make Headway. In L'Atelier, the BNP Paribas Group's, 2013. [http://www.atelier.net/en/trends/articles/internet-things-needs-standards-order-make-headway\\_425738](http://www.atelier.net/en/trends/articles/internet-things-needs-standards-order-make-headway_425738)
5. Sundmaecker H., Guillemin P., Woelfflé S. Vision and Challenges for Realising the Internet of Things. In CERP-IoT Cluster of European Research Projects of the Internet of Things, European Commission (2009), 12-13.
6. Rumpel A., Meissner K. Requirements-Driven Quality Modeling and Evaluation in Web Mashups In Quality of Information and Communications Technology (QUATIC), 2012, Eighth International Conference on the, IEEE Press (2012), 319-322.
7. L. Richardson and S. Ruby. RESTful Web Services. O'Reilly Media, Inc., 2007.
8. D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mash ups in the web of things, " in Proceedings of the 6th international conference on Networked sensing systems, Pittsburgh, Pennsylvania, USA, 2009, pp. 196-199.
9. D. Zhiquan, Y. Nan, C. Bo, C. Junliang, "Data Mashup in the Internet of Things", Proc. International Conference on Computer Science and Network Technology (ICCSNT), pp. 948-952, December, 2011.
10. GeoThings, <http://geothings.io/>.
11. Fitbit, <http://www.fitbit.com/>.
12. openHAB, <http://www.openhab.org/>.
13. M. Blackstock, R. Lea, "IoT mashups with the WoTKit", Internet of Things (IOT) 2012 3rd International Conference on the IEEE, pp. 159-166, 2012.
14. Kaa, <https://www.kaaproject.org/>.
15. Predix, <https://www.predix.io/>.
16. Carriots, <https://www.carriots.com/>.
17. W. S., "Business process modeling improves administrative control," Automation, pp. 44-50, 1967.
18. A. Aluva, "Why DIY Concept Has Started Trending Among Indian Startups - Inc42 Media," 2015. [Online]. Available: <https://inc42.com/resources/startups-adopting-do-it-yourself-models-diy-concept-trending-among-indian-startups/>. [Accessed: 24-May-2016].
19. R. Kleinfeld, S. Steglich, L. Radziwonowicz, and C. Doukas, "Glue.Things: A Mashup Platform for Wiring the Internet of Things with the Internet of Services," *Proc. 5th Int. Work. Web Things*, pp. 16-21, 2014.
20. Roberto, "Introduction to Node RED | Sensetecnic," 2015. [Online]. Available: <http://developers.sensetecnic.com/article/introduction-to-node-red/>. [Accessed: 19-Apr-2016].
21. S. Heo, S. Woo, J. Im, and D. Kim, "IoT-MAP : IoT Mashup Application Platform for the Flexible IoT Ecosystem," in *5th International Conference on the Internet of Things (IoT)*, 2015, pp. 163-170.
22. C. Richardson, "Overview of POJO programming," 2006.
23. F. Pramudianto, C. A. Kamienski, E. Souto, F. Borelli, and L. L. Gomes, "IoTLink : An Internet of Things Prototyping Toolkit," in *11th International Conference on Ubiquitous Intelligence & Computing*, 2014, pp. 1-9.

24. H. Nguyen, M. Quoc, and M. Serrano, "Super Stream Collider-Linked Stream Mashups for Everyone," in *Semantic Web Challenge co-located with ISWC2012*, 2012.
25. D. Carlson, M. Mögerle, M. Pagel, S. Verma, and D. S. Rosenblum, "Ambient Flow: A visual approach for orchestrating smart devices in the internet of things," in *5th International Conference on the Internet of Things (IoT)*, 2015.
26. N. Kefalakis, J. Soldatos, A. Anagnostopoulos, and P. Dimitropoulos, "A Visual Paradigm for IoT Solutions Development," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9001, pp. 169-182, 2015.
27. M. M. Burnett and B. a. Myers, "Future of end-user software engineering: beyond the silos," *Proc. Futur. Softw. Eng. - FOSE 2014*, no. April 2016, pp. 201-211, 2014.
28. Hyemin Lee, Dongig Sin, Eunsoo Park, Injung Hwang, Gyeonghwam Hong and Dongkun Shin, "Open software platform for companion IoT devices", 2017 IEEE International Conference on Consumer Electronics (ICCE).
29. IoTivity, [www.iotivit.org](http://www.iotivit.org).
30. oneM2M-Standards for M2M and the Internet of Things, [www.onem2m.org](http://www.onem2m.org).
31. J. Swetina, "Toward a Standardized Common M2M Service Layer Platform: Introduction to oneM2M", *IEEE Wireless Commun.*, vol. 21, no. 3, pp. 20-26, June 2014.
32. Cengiz Gezer, Erhan Taskin, "An overview of oneM2M standard", *Signal Processing and Communication Application Conference (SIU)*, 2016 24th.
33. OpenHAB, "openHAB - empowering the smart home," 2013. [Online]. Available: <https://github.com/openhab/openhab>.
34. OpenIoT, <https://github.com/OpenIoT/openiot>.
35. J. Hosek, P. Masek, D. Kovac, M. Ries, F. Kröpfl, "Universal Smart Energy Communication Platform", *Proceedings of 1st International Conference on Intelligent Green Building and Smart Grid (IGBSG 2014)*, pp. 134-137, 2014.
36. Arduino, <http://www.arduino.cc/>.
37. Raspberry Pi, <http://www.raspberrypi.org/>.
38. Intel Edison, [https://en.wikipedia.org/wiki/Intel\\_Edison](https://en.wikipedia.org/wiki/Intel_Edison).
39. M. Kovatsch, "CoAP for the Web of Things: From Tiny Resource-constrained Devices to the Web Browser," in *4th International Workshop on the Web of Things (WoT 2013)*, 2013.
40. M. Butcher, "REST Without JSON: The Future of IoT Protocols - DZone IoT," 2015. [Online]. Available: <https://dzone.com/articles/json-http-and-the-future-of-iot-protocols>. [Accessed: 23-May-2016].
41. Tensorflow, <https://www.tensorflow.org/>.
42. WEKA, <https://weka.wikispaces.com/>.
43. DEEPLARNING4J, <https://deeplearning4j.org/index.html>.
44. Lasagne, <https://github.com/Lasagne/Lasagne>.
45. Keras, <https://keras.io/>.
46. Muhammad.Sohail. Khan, "Enhanced IoT Composition Architecture based on DIY Business Process Modeling Approach".
47. ASHRAE. 2010. ANSI/ASHRAE Standard 55-2010. Atlanta: American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.
48. Fanger, P.O. 1970. -Analysis and Applications in Environmental Engineering. McGraw-Hill Book Company, New York.
49. L. Ma, H. Zhu, G. Nallamothu, B. Ryu, and Z. Zhang, "Impact of linear regression on time synchronization accuracy and energy consumption for Wireless Sensor Networks," in *IEEE Military Communications Conference (MILCOM)*, 2008.
50. A. Wemhoff, "Calibration of hvac equipment pid coefficients for energy conservation," *Energy and Buildings*, vol. 45, no. 0, pp. 60 - 66, 2012.
51. J. hong Yang and X. yan Bi, "High-precision temperature control system based on pid

- algorithm," in Computer Application and System Modeling (ICCASM), 2010 International Conference on, vol. 12, Oct 2010.
52. A. Giantomassi, F. Ferracuti, S. Iarlori, S. Longhi, A. Fonti, and G. Comodi, "Kernel canonical variate analysis based management system for monitoring and diagnosing smart homes," 2014, pp. 1432-1439.
  53. S. Murakami, S. Kato, and T. Kim, "Coupled simulation of convection, radiation, and hvac control for attaining a given pmv value," Building and Environment, vol. 36, no. 6, pp. 701 - 709, 2001, building and Environmental Performance Simulation:Current State and Future Issues.
  54. J. Cigler, S. Privara, Z. Vana, E. Zacekova, and L. Ferkl, "Optimization of predicted mean vote index within model predictive control framework: Computationally tractable solution," Energy and Buildings, vol. 52, no. 0, pp. 39 - 49, 2012.
  55. --, "Fuzzy logic home energy consumption modeling for residential photovoltaic plant sizing in the new italian scenario," Energy, vol. 74, no. 0, pp. 359 - 367, 2014.
  56. libcoap, <https://github.com/obgm/libcoap>.
  57. jFuzzyLogic, <http://jfuzzylogic.sourceforge.net/html/index.html>.