Towards a Comfortable, Energy-Efficient Office using a Publish-Subscribe Pattern in an Internet of Things Environment

LL Butgereit[1,2]
[1]Nelson Mandela Metropolitan University and [2]CSIR Meraka Institute
South Africa
laurie.butgereit(at)nmmu.ac.za
lbutgereit(at)meraka.org.za

AC Smith[2,3]
[3]University of South Africa and [2]CSIR Meraka Institute
South Africa
acsmith(at)csir.co.za

T Thomson
CSIR Meraka Institute
South Africa
connect(at)thanethomson.com

Abstract

The modern office is maintained by air conditioners, heaters, lights, smoke detectors, thermostats and other devices. Manufacturers are continually improving these various devices and adding more features – especially Internet connectivity. These new devices are often called *smart* devices. The Internet of Things is the new paradigm of more and more physical objects being connected to the Internet. As these objects begin interacting, mechanisms need to be in place to route data messages from the many objects to the correct applications which process that data. For example, the data from the thermostat in Office A must not be routed to the application controlling the temperature in Office B. One such mechanism is the publish-subscribe pattern often just called the pub-sub pattern. The pub-sub pattern enables applications to subscribe to data from certain resources. When a resource publishes data, the surrounding platform ensures that the data is forwarded to the correct subscribers. This paper describes an implementation of the pub-sub pattern specifically for an Internet of Things platform which operated at four levels – sensors (and actuator), Supervisors, Middleware, and application. This platform was specifically instantiated to control a typical office meeting room. The instantiation monitors the state of the doors and windows (open or closed), whether there are people in the room, the current temperature in the room, and the current electrical consumption. It then makes intelligent decisions as to how best to control the air conditioner in the meeting room.

Keywords

IoT, pub-sub

1 Introduction

The modern office consists of typical digital devices such as computers, fax machines, and telephones and also includes devices which are not primarily thought of as digital devices. These unseen devices consist of items such as air conditioners, heaters, smoke detectors, and lights. However, these devices are becoming more and more digital as manufacturers add features to them. In adding features, these devices start generating data and responding to data.

Beyond the office, the world is full of data. A simple walk down the street could overwhelm a person with millions of bits of data if there was not some sort of discrimination mechanism to filter this information. Sometimes this human discrimination mechanism gets distorted. The prime example of this is when a person buys a new car and then begins to notice that same make and model of that car everywhere in traffic. The prevalence of that make and model car does not increase substantially; the person merely notices the make and model car more often.

The Internet of Things is also full of data. As more and more physical objects become Internet enabled, the amount of data increases. For applications which are monitoring these objects or subsets thereof, a mechanism is needed to enable the applications to only receive data from objects in which they have an interest. One such programming pattern which has been implemented to solve this problem is the publish-subscribe pattern (often called the pub-sub pattern).

In a publish-subscribe implementation, subscribers have the ability to express an interest in a particular event and they are subsequently notified by a publisher when the event occurs (Eugster, Felber et al. 2003, Huang, Garcia-Molina 2004, Tarkoma 2012). This paper describes work in implementing a pub-sub pattern for the Internet of Things linking together four levels of components from sensors (or actuators), a supervisor device, a middleware router, and an application.

2 Background

Since its inception, the Internet has grown from a few inter-connected computer nodes to a network servicing more than a billion users (Kopetz 2011). Physical objects are being enhanced with electronic components giving the objects the capability to interact with the Internet. This connection of physical things to the Internet makes it possible to access remote sensor data and to control the physical world from a distance (Kopetz 2011).

As more physical objects are connected to the Internet, scalability becomes increasingly important. As billions of objects start communicating using the Internet of Things, scalability mechanisms which enable high volume communication become more important (Uckelmann, Harrison et al. 2011).

3 Research Question and Methodology

The research question addressed with this paper is “Can the pub-sub pattern be implemented for an Internet of Things platform which controls the level of human comfort in an office environment?” Here, the paper investigates a phenomenon which is the result of human activity and also under the control of humans as opposed to activities that are beyond human control. Vaishnavi and Kuechler (2013) indicate that the Design Science Research methodology is well suited for investigating a phenomenon as described here. Purao (2013) also indicates that the goal of the Design Science Research methodology is to change reality through the creation of artifacts in order to address a perceived problem. The problem addressed here is the use of a pub-sub pattern used in conjunction with the Internet of Things in order to control certain aspects of an office environment. From the preceding, it seems that the Design Science Research methodology would suit the current research project well. Therefore, the Design Science research methodology is used in conducting the current research.

The application of the Design Science Research methodology prescribes that an artifact be developed. The artifact could be a construct, a model, a method, and/or an instantiation (March, Smith 1995, Hevner, March et al. 2004). Constructs consist of vocabulary and symbols, whereas, models are abstractions and representations. Methods are algorithms and practices, whilst instantiations are implementations or prototypes of models and/or methods.

Hevner et. al. (2004) state that the Design Science Research methodology is a design process which is both incremental as well as iterative. What makes the Design Science Research methodology attractive for certain types of research questions is that this methodology makes provision that not all the required knowledge for answering the research question is known at the onset of a project. As described by Vaishnavi & Kuechler (2013), this methodology acknowledges that additional information is gathered during each iteration. Here, an iteration typically consists of three phases, and these are the suggestion phase, the development phase, and the evaluation phase. An additional phase is relevant at the onset of the research, and this phase is called “awareness of the problem”. In addition, a final phase is executed at the end of the final iteration, and this phase is called “conclusion”. Whereas a full iteration comprises of these five phases, not all the phases need to be executed during all the iterations; some iterations may terminate immediately after either the development or evaluation phases have been concluded. Upon such termination, knowledge gained from the current iteration is added to the existing (and incomplete) knowledge base and this process is called “circumscription”.

For the scope of this research project, both a model of how the pub-sub pattern could be used with Internet of Things was created and an instantiation or implementation of that model across four levels of interaction.

Kuechler, Vaishnavi, and Petter state that Design Science Research is also called “improvement research” (Vaishnavi, Kuechler 2007, p. 32) which emphasises the problem solving and improvement nature of the research. The set of five steps mentioned above that comprise awareness, suggestion, development, evaluation, and conclusion, is also known as the General Design Cycle.

All Design Science Research begins with an awareness of a problem. This awareness can be created by reading literature, conversations with colleagues, or personal experience. The suggestions on how to solve this problem can also come from existing literature or can come from conversations with colleagues. These suggestions are then investigated in detail and actually developed. The resulting artifact is subsequently evaluated to see if it solves the problem. Conclusions can then be drawn.

It is important to note that the General Design Cycle is, in fact, a *cycle.* The fives steps can be iterated numerous times, with the design usually improving in each cycle.

4 Architecture

An overview of the architecture is depicted in Figure 1. At the lowest level of the architecture are a number of sensors and actuators which communicate with the physical world. These include a passive infrared (PIR) detector, temperature sensor, current sensor, door and window sensors, and an actuator in the form of an air conditioner. All the sensors send their data to an Arduino Uno circuit (Barrett 2012), whilst the same circuit controls the actuator. In turn, the Arduino board communicates with the Raspberry Pi circuit (Barnes 2014) via a USB cable. Python programming language software executing on the Raspberry Pi circuit implements the pub-sub pattern. This software is called the *Supervisor*. Supervisors are designed to communicate with the *Middleware* using either a wired network or wireless network (depending on the capabilities of the particular Raspberry Pi circuit employed in the design). The Middleware executes on a host computer which, in turn, routes messages to the correct application. Communication between the Supervisor and the Middleware is possible using standard Internet communication layer technology such as a wired network (CAT5 or CAT6) or a wireless network such as WiFi. Our implementation makes use of a wired network. Although this architecture makes provision for multiple supervisors, this research project only considers a single supervisor. RESTful HTTP is the higher level protocol that links the Raspberry Pi, the Middleware, and the applications. Communication between the Arduino Uno circuit and the Raspberry Pi circuit is supported by the open Firmata protocol (Margolis, Weldin 2011).

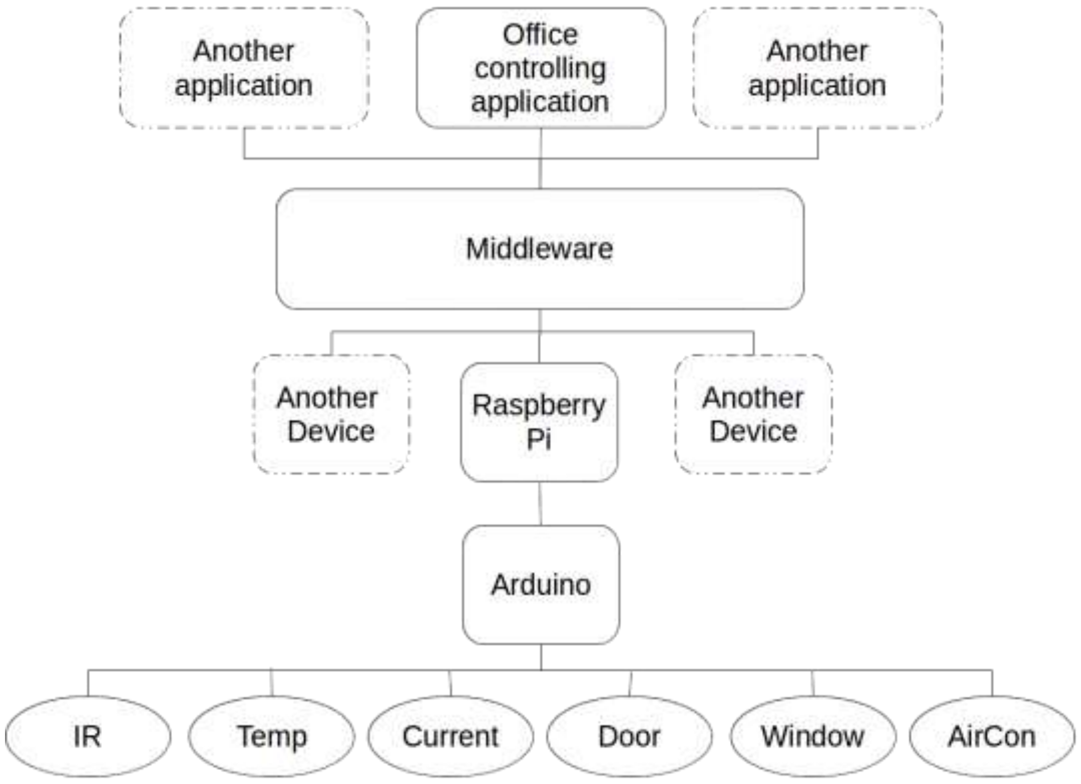

Figure 1: Architecture

In our implementation, all HTTP messages for pub-sub are encoded as JSON (JavaScript Object Notation) messages. It is important to note that the messages for resource discovery (which are not necessarily the subject of this paper) are in CoRE Link format (Shelby 2012) rather than JSON. The actual format of these JSON messages constitute the bulk of this paper.

As more physical objects become connected to the Internet, the imperative to move from IPv4 (Internet Protocol version 4) to IPv6 (Internet Protocol version 6) becomes stronger. IPv6 (Stallings 1996) uses 128-bit addressing, thereby providing for 2^{128} uniquely identifiable devices as opposed to the alternative IPv4 addressing which uses 32-bit addressing and consequently allowing for only 2^{32} uniquely identifiable devices (Hain, Huston 2005). In this model and implementation, IPv6 is only used for communication between the Middleware and the Supervisor. The remaining portion runs on IPv4.

5 Enrolment

When supervisors and/or applications wish to utilise the Middleware, it is necessary to first enrol with the platform. This enrolment process requires information including a unique identifier, and the type of the object. Of primary importance, however, is that the enrolment process includes the IP address (either IPv4 or IPv6) of the object and a port number. Depending on the object type, additional information could be included. For example, in the case of software applications an additional context could be provided. A sample message could look like:

```
{
      "name" : "name of the object",
      "hostAddress" : "host address and optional port number:,
      "uid" : "unique identifier",
      "context" : "optional context of use by software applications",
      "type" : "the type of the object"
}
```

This information is sent by either the Supervisor or the application to the Middleware using RESTful PUT.

6 Resource Discovery

In order for the pub-sub pattern to work, it must be preceded by a discovery of the available resources. For example, in Figure 1, if the application shown needs to subscribe to a published value on a sensor, it must first discover that sensor. There are a number of mechanisms which could be used for discovering resources (such as sensors and actuators) (Shelby 2010), but the use of a well-known core file also known as CoRE (Constrained RESTful Environments) Link format (Shelby 2012) was chosed for this research.

In the case of an implementation (such as this one) that uses the Constrained RESTful Environment Link format, each Supervisor publishes a well-known core file. This file is an aggregation of the devices which the Supervisor monitors and actuates. The Middleware then aggregates all well-known core files from each supervisor. In addition, at a higher level, applications can also publish their own well-known cores which are also aggregated at the Middleware level. An example of an extract from a well-known core file is as follows:

```
<http://[xxxx:xxxx:xxxx:x::312]:8104/window>;rt="window-
string";uid="20001234";title="Matrix boardroom window state as o/c",

<http://[xxxx:xxxx:xxxx:x::312]:8104/temperature>;rt="temp-
float";uid="20001234";title="Matrix boardroom temperature in Deg C",

<http://[xxxx:xxxx:xxxx:x::312]:8104/ampere>;rt="ampere-
float";uid="20001234";title="Matrix   boardroom   airconditioner   power
consumption"
```

7 Subscribe and Unsubscribe

When an application or other entity wishes to subscribe to a resource (assuming that discovery has already occurred), a subscription request is sent from the application to the Middleware via a RESTful PUT message. The JSON subscription messages adhere to the following format:

```
{
        "publisher"       : "publisher identifier",
        "subscriber"      : "subscriber identifier"
 }
```

Here, the strings "publisher identifier" and "subscriber identifier" represent identifying strings of the publisher and subscriber respectfully. Other security messages unrelated to the pub-sub pattern are implemented to keep track of the IP address (IPv4 or IPv6 as appropriate) in order to send and identify subsequent messages.

Through previous discovery requests, the Middleware knows which Supervisor manages the sensor indicated by the publisher field and forwards the identical message to the Supervisor. In the model, in order to unsubscribe from a resource or sensor, the application must send the identical JSON request as an HTTP DELETE message. However, in the actual implementation of this feature, it was found that the Java implementation of HttpURLConnection does not support a payload with the DELETE request and a modified format was used in the instantiation by sending all the required data as HTTP query parameters.

8 Data

In the current implementation, the Supervisor continually monitors data received from the sensors. When appropriate, the supervisor forwards this data to the Middleware as a JSON message of the following format:

```
{
    “resource”  : “publisher identifier”,
    “values”    : [
                      { “timestamp”   : “date value” },
                      { “value”       : “data value” } …
                  ]
 }
```

where “publisher identifier” is the same publisher indicated in the subscription. Any number of timestamp/value pairs can be included in the values array. This bundling of data would be appropriate when Supervisors were offline for periods of time. When the Middleware receives such a message, it forwards it to as many applications as have subscribed to the particular sensor. In other words, data is sent once from the sensor to the Supervisor and once from the Supervisor to the Middleware. Then the data is forwarded multiple times to the various applications that have subscribed to the particular sensor. In addition, the Middleware stores all data on a transaction file for later analysis if necessary.

9 Implementation

For the actual implementation for this research project, an office meeting room was instrumented with multiple sensors and one actuator. Sensors were mounted on the door and window which could indicate if the door or window changed state from open to closed or from closed to open. A passive IR sensor was installed to determine if there were people in the room. There were also a room temperature sensor and an electrical current sensor on the air conditioner unit. In addition, there was an actuator which controlled the output of the air conditioner. The air conditioner could be instructed to turn on, to turn off, to increase output, and to decrease output. However, a heater was not implemented in this research project.

9.1 Application

In this particular implementation, the application subscribed to the change of state in a door and in a window. That means if the state of the door or the state of the window changed from open to closed or from closed to open, the application received that information. The application also subscribed to the passive IR sensor to indicate if there were people in the room. In addition, the application subscribed to the temperature in the office meeting room and periodically received this information. At the time of authoring this paper the current sensor was not used. The application implemented a number of rules such as:

1. If the door or window was open, then the air conditioner unit would be turned off.
2. If the door and window were closed, and there were people in the room, then the air conditioner would be turned on.
3. If the temperature was above a certain threshold, and there were people in the room, and the door and window were closed, then the air conditioner would increase output.
4. If the temperature was below a certain threshold, and there were people in the room, and the door and window were closed, then the air conditioner would decrease output.

9.2 Supervisor

Although the Raspberry Pi Model B circuit contains a powerful processor along with USB ports, high definition video output capabilities, solid state memory, and Ethernet connectivity, it lacks robust input/output capabilities with which to interface with experimental electronic sensing and control circuits. An often used companion for the Raspberry Pi circuit is the Arduino series of electronic circuits. The Arduino circuits, and specifically the Arduino Uno series, are designed and developed as open technology experimental circuits. These are ideal for novice electronic enthusiasts who want to connect devices in the physical domain to software processes executing in the digital domain. Although the Arduino circuit lacks the processing power, USB ports, Internet connectivity and solid state memory capacity of the Raspberry Pi, it excels at simplicity, convenience, cost, and robustness. These three attributes make the Arduino Uno circuit an ideal companion to the Raspberry Pi circuit when the digital and physical domains have to be interconnected. For the current implementation of the Supervisor, the Raspberry Pi and Arduino Uno circuit combination as described here have been used.

9.2.1 Raspberry Pi Circuit

In the current implementation, the Raspberry Pi circuit executes complicated software routines whereas the Arduino Uno circuit executes relatively mundane routines. Although these routines executing on the Arduino Uno circuit may not be as sophisticated as those executing on the Raspberry Pi circuit, they nevertheless provide the interface between the physical and digital domains which is so important to this research project.

The processes executing on the Raspberry Pi circuit rely on a version of the Debian operating system for support, such as providing access to the Ethernet and USB ports. This version is called Raspbian. All of the custom software executing on the Raspberry Pi circuit were written in the Python interpreted programming language. This language is preinstalled with the Raspbian operating system and is simple to use.

Two software modules are core to the operation of the Supervisor. The first of these is the module which interfaces with the process executing on the Arduino Uno circuit. Using this module, the status of the Arduino Uno circuit's hardware pins can be monitored and controlled. How these pins are used is described in the paragraphs that deal with the Arduino Uno circuit. Depending on how frequently the overall system needs to be informed of changes in the physical domain, the software can interrogate the Arduino Uno circuit at a wide range of timing intervals. In the current implementation, the status of the Arduino Uno circuit's hardware pins are updated at a rate of approximately once a second. Although this rate is not usually required for sensing the status of doors and windows, or determining the temperature in the room, it is very useful in determining the instantaneous current consumed by the air conditioner in the room. These currents can fluctuate widely within the span of a few seconds and it is worthwhile to capture these fluctuations at a resolution of one second intervals. The second software module executing on the Raspberry Pi circuit is responsible for communicating with the Middleware. Communication is accomplished using the Constrained RESTful Environments Link format and JSON formatted messages. Physical communication between the Supervisor and the Middleware is achieved using the Ethernet interface which is available on the Raspberry Pi circuit. An alternative physical communication channel is by means of a WiFi radio, but this research project did not implement this option. The following subsection describes the Arduino Uno circuit which is used in tandem with the Raspberry Pi circuit to create a Supervisor hardware module.

9.2.2 Arduino Uno circuit

A Raspberry Pi circuit is not fully capable of interfacing to the physical domain without the aid of additional electronic circuitry. It is for this reason that the Arduino Uno circuit is used in conjunction with the Raspberry Pi circuit. Together, these two circuits constitute the hardware framework of the Supervisor. A system architect can specify which sensors and actuators should be integrated with the Supervisor framework. In the current design, two separate sensors for a door and a window, a passive IR sensor, a temperature sensor, and a sensor to measure the flow of electrical current to the air conditioner located in the meeting room are implemented. Although the Arduino Uno circuit does not host a powerful processor such as contained in the Raspberry Pi circuit, it nevertheless has a processor suitable for controlling and sensing connected devices. The same processor also executes software which communicates with those executing on the Raspberry Pi circuit. Communication between these two circuits follows a protocol implemented by the open standard Firmata software suite. This software suite was developed by a volunteer software community and is available for use on a wide variety of processor dependant hardware platforms, of which the Arduino Uno circuit is one. Using the Firmata suite on the Arduino Uno circuit, commands can be received from the Raspberry Pi circuit and results can in turn be relayed back to the Raspberry Pi circuit from the Arduino Uno circuit.

As used in this research project, each of the door and window sensors is connected to one of two input pins respectively on the Arduino Uno circuit. Because the door and window sensors are in the form of reed switches, additional biasing resistors were also attached to the two respective input pins. Without these resistors, and while the reed switches are in an open state, spurious electrical signals might cause erratic changes in the sensed pin; the addition of the resistors eliminates this problem. Both the door and windows sensors are activated by the presence of a magnetic field. To sense whether a door is open, a reed switch is attached at the top of the door frame with this position being on the side opposite to the door hinges. A permanent magnet is in turn attached to the door in a position corresponding to the reed switch. The result is that the reed switch is activated by the permanent magnet when the door is closed, and the reed switch opens again when the door opens. Detecting the status of the window is similar to that of the door. Here, the magnet is attached to the top of the window and the reed switch to the top of the window frame in a corresponding position.

Temperature is sensed using a dedicated sensor procured for this purpose. This sensor has three electrical connections and presents the current ambient temperature in degrees

Celsius. Two of these connections supply power to the sensor, with the third connection being assigned to providing data representing the temperature.

It is not only the temperature sensor that requires a power source; the passive IR sensor also contains its own electronic circuit which must be powered via two wires. As is the case for the temperature sensor, power is supplied by the Arduino Uno circuit. Also similar to the temperature sensor, a third electrical connection provides data that reflect the presence or absence of one or more persons in the meeting room. The data thus provided is maintained on the third electrical connection for a period of approximately 500 milliseconds and can be described as follows: When no person has been detected for 500 milliseconds, the data is set to a binary “0”. However, when a person is detected, the data is set to a binary “1” and remains in that state for 500 milliseconds. After this time has expired the data will be set to a binary 0 until such time as a person is detected again.

Detecting the electrical current that flows to the air conditioner is somewhat more complicated than detecting the binary states of the sensors described above. To detect the electrical current, a device known as a “current transformer” is attached around the “live” wire leading to the air conditioner. A resistor is connected *in situ* and across the current transformer. The result is a varying analogue voltage that provides a direct indication of the current flowing through the live wire. Unfortunately this simple configuration does not cater for a phase difference between the voltage and the current flow in the wire being monitored. However, for the purposes of this research project, this small inaccuracy is acceptable. An analogue sensing pin on the Arduino Uno circuit provides a means of measuring the changing analogue signal.

The final attachment to the Arduino Uno circuit is an infrared emitting diode with which the air conditioner can be controlled. This diode is placed in close proximity to the air conditioner’s own infrared signal receiver, thereby allowing the Arduino Uno circuit to control some of the functions of the air conditioner in lieu of the usual hand controller.

10 Results

A number of experiments were conducted while the meeting room was in use. Specific problems which were encountered included:

1. The air conditioner in the office was mounted high on the wall and, initially, the temperature sensor was mounted at that same height. This location was not ideal and the sensor was subsequently moved. Initially, when the air conditioner expelled cold air, the cold air descended below the temperature sensor. This meant that during our original implementation, the application continually received

temperatures which were high (hot air rises) and kept increasing the output of the air conditioner unnecessarily. However, the people in the meeting room were experiencing very cold temperatures since the cold air produced by the air conditioner settled at lower levels. Later configurations placed the temperature sensor at desk height.

2. Passive IR sensors operate by detecting signal changes as a heat source moves across adjacent detection zones. These zones are determined by a Fresnel lens mounted in front of an IR detector. Body heat generated by humans is usually sufficient to trigger a passive IR sensor. Using such sensors, it is usually possible to determine if one or more persons are within a room as long as at least one person does not remain stationary. If a person is seated in a high backed chair and turned away from the sensor, the sensor is not able to detect the radiated heat through the chair back. In some cases, if the person was above average height and his or her shoulders and head cleared the high chair back, then the person could be detected. Shorter people were often not detected. For the scope of this research, this problem was not satisfactorily solved.

The solution to problem #1 above, however, is an example of the advantage of the General Design Cycle: awareness, suggestion, development, evaluation, and conclusion. By evaluating the design numerous times and cycling over the steps, the design could be improved. For example, on the actuation side:

1. The controlling application could turn the air conditioner on and off appropriately (taking into account the two problems indicated above).
2. The controlling application could increase or decrease the output of the air conditioner appropriately (again, taking into account the two problems indicated above).

The primary scope of this research, however, was to determine if the pub-sub pattern was suitable for such an environment which attempted to control the comfort of an office meeting room while, at the same time, being energy efficient. In this respect, the pub-sub mechanism worked properly and removed the requirement of wasteful polling to obtain data from the sensors.

11 Conclusion

As more tangible objects become Internet connected, the Internet of Things grows. Mechanisms need to be put in place to be able to easily monitor the data which are being generated by this vast collection of smart objects. This paper investigates whether the pub-sub model is suitable for an Internet of Things platform which operates at four levels (sensors or actuators, Supervisor, Middleware, and application). A model is created for this interaction. The model was subsequently instantiated using HTTP with all HTTP messages being in the format of JSON messages.

Although this paper primarily investigates the use of the pub-sub pattern, it is important to note that there are two supporting mechanisms. First, there is an enrolment/unenrolment mechanism allowing subscribers to join and leave the platform. The enrolment process also provides important additional information about the object. Second, there is also a discovery mechanism which allows subscribers to look for potential publishers of information.

As the word “towards” in the title of this paper indicates, the work reported here is the first of multiple anticipated design cycles. Some of the design challenges we faced are stated in the results section, but other challenges remain. For instance, we are investigating mechanisms by which we can determine the number of persons in the room as well as get an indication of their activities within the room. We then hope to find a correlation between the energy consumption and the activities taking place within the room.

Acknowledgement

The authors would like to thank the valuable comments and suggestions of an anonymous reviewer and hope to address these appropriately in a subsequent paper.

12 References

BARNES, R., 2014. *Raspberry Pi - the Complete Manual.* Imagine Publishing Ltd.

BARRETT, S.F., 2012. Arduino Microcontroller: Processing for Everyone! *Synthesis Lectures on Digital Circuits and Systems,* **7**(2), pp. 1-371.

EUGSTER, P.T., FELBER, P.A., GUERRAOUI, R. and KERMARREC, A., 2003. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR),* **35**(2), pp. 114-131.

HAIN, T. and HUSTON, G., 2005. A pragmatic report on IPv4 address space consumption. *The Internet Protocol Journal,* **8**(3), pp. 2-19.

HEVNER, A.R., MARCH, S.T., PARK, J. and RAM, S., 2004. Design science in information systems research. *Management information systems quarterly,* **28**(1), pp. 75-106.

HUANG, Y. and GARCIA-MOLINA, H., 2004. Publish/subscribe in a mobile environment. *Wireless Networks,* **10**(6), pp. 643-652.

KOPETZ, H., 2011. Internet of things. *Real-Time Systems.* Springer, pp. 307-323.

MARCH, S.T. and SMITH, G.F., 1995. Design and natural science research on information technology. *Decision Support Systems,* **15**(4), pp. 251-266.

MARGOLIS, M. and WELDIN, N.R., 2011. *Arduino Cookbook.* O'Reilly Media, Inc.

PURAO, S., 2013. Truth or dare: The ontology question in design science research. *Journal of Database Management (JDM),* **24**(3), pp. 51-66.

SHELBY, Z., 2012. Constrained RESTful Environments (CoRE) Link Format.

SHELBY, Z., 2010. Embedded web services. *Wireless Communications, IEEE,* **17**(6), pp. 52-57.

STALLINGS, W., 1996. IPv6: the new Internet protocol. *Communications Magazine, IEEE,* **34**(7), pp. 96-108.

TARKOMA, S., 2012. *Publish/subscribe Systems: Design and Principles.* Wiley.com.

UCKELMANN, D., HARRISON, M. and MICHAHELLES, F., 2011. *Architecting the internet of things.* Springer.

VAISHNAVI, V. and KUECHLER, W., 2007. *Design science research methods and patterns: innovating information and communication technology.* CRC Press.

VAISNAVI, V. and KUECHLER, W., 2013-last update, Design Science research in information systems. Available: http://www.desrist.org/design-research-in-information-systemsJune 23, 2013].