
Proceedings Paper:
Cameron, C and Li, K (2018) Low-Cost, Extensible and Open Source Home Automation Framework. In: Li, K, Fei, M, Du, D, Yang, Z and Yang, D, (eds.) Intelligent Computing and Internet of Things. IMIOT 2018: International Conference on Intelligent Manufacturing and Internet of Things and ICSEE 2018: 5th International Conference on Computing for Sustainable Energy and Environment, 21-23 Sep 2018, Chongqing, China. Springer, pp. 408-418. ISBN: 978-981-13-2383-6. ISSN: 1865-0929.

https://doi.org/10.1007/978-981-13-2384-3_38

Low-Cost, Extensible and Open Source Home Automation Framework

Ché Cameron[1] and Kang Li[2]

[1] Queens University Belfast, University Road, Belfast BT7 1NN
[2] University of Leeds, Leeds, LS2 9JT

Abstract. A framework for home IoT development and automation is proposed, and the details of a low cost demonstration system synthesised using this approach are recounted. Details are given on the roles of each component and the functionality achieved by the system - control of central heating, temperature monitoring and finger print based access control. The performance of the system and its data acquisition abilities are reflected upon to suggest future development work.

Keywords: Home Automation, IoT, Open-Source, Low Cost

1 Introduction

The field of IoT has been the topic of much discussion and development over the past few years, and has arguably suffered from its own hype. The stunning array of low-cost sensors, actuators and controllers presents a panoply of possible applications - and in so doing, runs the risk of an over-abundance of choice. This is reflected in the lack of harmonisation or standardisation throughout the field [1].

However, some standard tools and techniques have started to emerge which are helping to increase interoperability between ecosystems and devices [2] - an important aspect of the interconnected world which is strived for. In this fluid and constantly developing landscape, a strong strategy for platform development focuses upon wide compatibility, technological agnosticism and quick reconfigurability [3].

Useful technology can be defined by the impact it has upon our lives, and the home environment presents a prime opportunity for this impact. Some schools of architecture consider the home as a *machine for living* and in this vein, the most important functions of that machine could be considered as suitable targets for IoT technologies.

The temperature of a dwelling is one of the most important aspects of comfort (and even safety, in extreme conditions) and usually consumes the largest proportion of the total energy expended in the home - both in empirical terms and in cost. Central heating control is a natural starting point for many home IoT systems, which often build out their functionality from this foundation. [4] Proactive energy management of a home plays a large role in setting household running costs and influences environmental impact - and is greatly assisted and optimised through the use of smart interconnected systems.

The selection of other applications could be informed by considering the interactions that a person has with their home - switching lights or appliances on and off, consuming entertainment, entering and exiting the abode, security, maintenance, repair and cleaning, to specify only a few.

The choice of 'framework' as a description for the presented work is a deliberate choice to underscore the flexibility of this suite of technologies, as such, the system is not a fixed platform but rather a demonstration of what can be achieved with flexible, low cost components, along with suggestions for future development. The details of data processing and analytics, including intelligent event detection are a topic for future work and will only be briefly touched upon.

A selection of available IoT platforms will be overviewed (with a focus on Open-Source) in Section 2, then this paper will introduce the framework in Section 3, including the details of the software packages, then go on to describe the demonstration system in Section 4, with information about the implementation in hardware and software. Section 5 offers some observations and thoughts on the results of using this framework in a practical setting, and the work is briefly concluded in Section 6.

2 IoT Platforms

The framework that will be presented here is more akin to an architecture than any individual products and so it is most fruitful to consider the major Home IoT ecosystems in use today rather than attempt to review the wide range of individual products available.

Amazon Alexa, Google Home and Apple Homekit, nest products and Samsung Smartthings make up a large segment of the market and give a fair representation of the shape of closed-source Home IoT technologies. They are characterised by expensive components, centralised servers for management and beautiful, highly developed interfaces that require minimal effort from the end-user to set up.

Initially, each company tried their best to create a closed ecosystem with a monopoly on compatible products, however consumer pressure is forcing them to gradually open their platforms to more devices [5]. The support that each offers is still far from universal, ultimately restrictive in what can be accomplished and largely pre-defines how the user interacts with their system.

Open source solutions are available for IoT and there are many, and ever-changing choices (as is custom in this arena). These solutions vary from presenting a complete management suite down to aggregation of control panels for disparate devices/systems. Some closed-source projects also aim to integrate disparate IoT ecosystems and generally monetise this effort by offering enhanced features for a price.

The general advantages of an open-source approach are transparency of operation (helping to safeguard security and privacy), the ability to customise the elements as desired and a reduction in cost. In a less tangible fashion, the community surrounding an open-source project is often passionate about what they are

building and using, which is often reflected in the level of help that is available from developers and peers.

Many of these platforms provide built-in support for a wide range of end devices and hubs, along with a community that continues to develop plug-ins. Without a specific application in mind, it is counter-productive to attempt to definitively rank the different projects - Table 1 shows a comparison of the important characteristics and features of a variety of systems.

PLATFORM	SET UP & CONFIGURATION	AUTOMATION / FEATURES	DEVICE INTEGRATION	PRICE
Calaos *[Open Source]*	GUI Configuration of plugins	Scripted Rules Engine, LUA Scripting	Wago PLC, multimedia, additional devices as developed	FREE
Domoticz *[Open Source]*	GUI Configuration of plugins, Auto-Detection of Devices	Simple Event Scripting, Notifications, Master/Slave Devices,	Wide Range of Plug-Ins	FREE
Home Assistant *[Open Source]*	YAML, Add-On Store, Text Config Files	Scripted Rules Engine (States Based), AppDaemon (Python),	1000+ Devices	FREE
openHAB *[Open Source]*	JVM Environment, Text Config Files, Limited GUI Config	Bindings API (text based), Experimental Rules Engine	Wide Range of Plug-Ins	FREE
OpenRemote *[Closed Source]*	Web Based UI Designer, Text Config Files, Scripting	Scripted Rules Engine (Drools)	Wide Range of Plug-Ins (variable quality and completeness)	Private: FREE, Commercial: $150
ThingsBoard *[Open Source]*	GUI Device Manager, Text Config Files	Rules Engine, Asset Tracking, Cluster Servers	MQTT, CoAP and HTTP with provision for legacy systems (eg. Modbus)	Basic:FREE, Advanced: $1999/year
iRule *[Closed Source]*	Web Based UI Builder	Device Commands only	Plug-Ins for Major EcoSystems, IR remote, NO custom plugin development	Basic: $49.99, Advanced: $99.99

Table 1. Comparison of IoT Platforms

The implementation of rules engines, scripting and programming languages is often proprietary (or at least non-standard). This means that basic, pre-defined or anticipated tasks are relatively easy, but a steep learning curve often presents itself upon any attempt to customise functionality outside of the designated development path. Therefore any large project is likely to generate significant amounts of code and effort that cannot be reused elsewhere. [6]

Each platform tends to have a particular philosophy or paradigm to champion, however therein lies a problem - the presupposition of applications, tasks and functionality. The framework described in Section 3 addresses this issue by connecting generalised tools with an open and flexible approach.

3 Home IoT Framework

3.1 Overview

This framework is composed of a networked hub, sensor/actor nodes and a web interface suitable for computer or mobile use.

The hub is responsible for node management, data handling (aggregation, storage and processing) and it integrates a web interface which may be optionally forwarded to the WAN for remote access. Programming and implementation is

achieved primarily through the flow-based Node-RED software on the hub, and inter-component communication is via the MQTT protocol making use of JSON encoding for data structures.

Physical nodes may be composed of any technologies that the user desires, in this case a mix of Arduino Nano microcontrollers equipped with NRFL2401+ radios and ESP32 integrated WiFi microcontrollers. Most modern homes incorporate a wireless network already, which this system can connect to - alternatively a separate air-gapped network or alternative radio technology can be used if desired. The intent is to demonstrate a low-cost, hardware agnostic approach to Home IoT which trades some ease of installation for large cost savings, fast development of applications and huge customisability.

The system is installed and operated in a three storey terraced house, and performs thermostatic heating control, logging of per-room temperature and access control via a fingerprint scanner - details of which are presented Section 4.

An important note should be made with respect to privacy - all data is handled and stored locally, owned by the user and never needs to traverse the internet unless explicitly instructed to do so. With growing awareness of the dangers posed by careless distribution of personal data [7] (reflected by the new GDPR regulations), a system which defaults to safeguarding the people who use it - empowering them with ownership and responsibility - is a step towards improving our collective attitude towards privacy versus convenience.

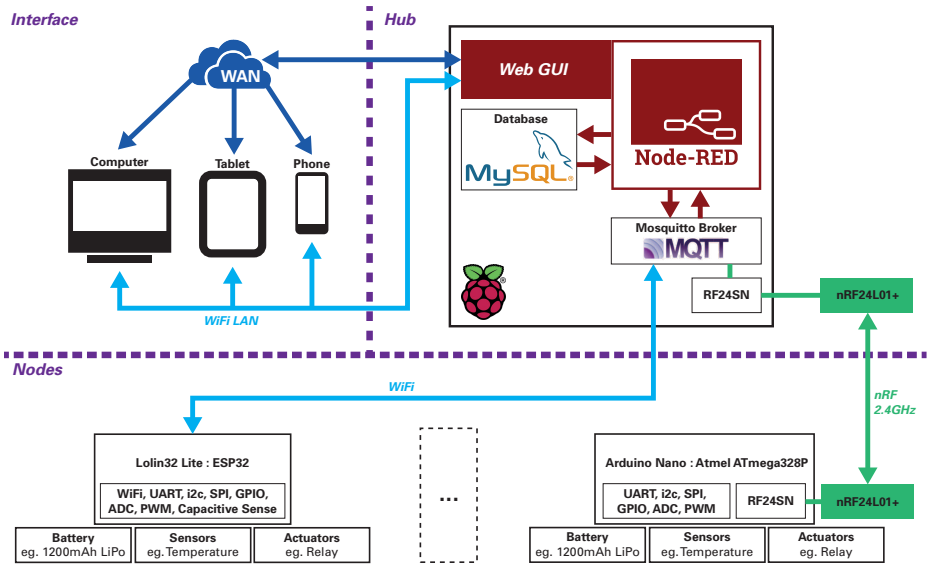

Fig. 1. Block Diagram of the Implemented Framework

3.2 Software

The presented cocktail of open-source software forms the essential components of this framework, providing for simple and fast development along with the ability to create complex and rich applications if so desired. The suite of software and its interrelations will be explained, along with some of the notable qualities of each component.

Communication - The nodes and hub communicate using the MQTT protocol which implements a topic based publish / subscribe model. The hub acts as a broker (which routes and buffers all network messages) and each client wishing to communicate connects to this hub. A single message is composed of a topic and a payload, each of which is a plaintext string of variable length.

The message topic defines its routing, with keywords separated by forward slashes. If a given client would like to receive messages they contact the broker with a subscribe message that indicates the topics that they are interested in. When a new message is received by the broker it will automatically be forwarded to all clients which have subscribed to that topic.

The hub makes use of the free and open-source broker mosquitto, and the only requirement placed upon nodes is that they must be able to communicate via MQTT - however this does not necessarily need to be accomplished on the node itself as the hub can implement a translation layer (note RF24SN in Figure 1). Complex data structures may be transferred using JSON encoding, which enables rich communication or lightweight messaging without the need for multiple communication protocols.

This choice of communication protocols and data structures reflects the flexible nature of the framework - using the most established lightweight standards maximises interoperability and minimises the burden upon hardware which enables the widest range of devices to be integrated into a synthesised system. [8]

Logic - Node management, business logic, data processing and display are all accomplished via Node-RED - a programming tool originally developed at IBM that was open-sourced in 2013 and continues to grow in popularity and functionality. This flow-based programming language runs on top of Node.js and enables unparalleled development speed and reconfigurability. Instead of traditional code based programming, the user drags and drops nodes on to a canvas and wires them together create functionality. In the most basic conception, nodes are classified into Input, Function and Output - with a rich set of features already included in the base installation.

Program operation is defined by messages which propagate through a given set of nodes - transformations of that message or actions triggered by it are defined by the logic that the user has wired into the program. Each set of nodes may include branches, conditional blocks or message routing to enhance the complexity of the program. By defining behaviour in this way, sections of a program naturally decouple and the relationship between different elements can

be clearly visually seen. The created 'codebase' is organised into one or more labelled tabs which are scoped individually.

In addition to this, tools are provided for data visualisation and user interaction in the form of widgets such as Charts, Gauges, Switches, Text Fields etc. A heavily customised and feature rich web interface can be produced with little to no programming experience, which vastly increases the potential users of such a system. There is a thriving open-source community constantly contributing to Node-RED, both with custom nodes and also full fledged programs (known as Flows) - the integration of which is often single click effort, again underscoring the ease-of-use for novice users.

This flow based paradigm is becoming an increasing focus in IoT research due to its quick reconfigurability and ease of use [9] - applications are not static and IoT implementations are made considerably more responsive and useful by enabling a larger proportion of the user base to modify their operation without needing to learn a complex programming language.

More advanced users can write custom functions in Javascript, create custom widgets using html/Angular code, group functionality into subflow nodes for easy re-use, and make completely new nodes with customised functionality. Scoped variables and objects are also available for buffering and communication of data within the program.

Node-RED is an excellent match for the requirements of a reconfigurable logic and data handling framework, and is being increasingly recognised as a powerful tool. [10] The fact that it is accessible to new users with excellent community support further enhances its strong position as the heart of a home automation framework.

Data Storage - Small sets of data can be stored as JSON encoded text files by Node-RED, however this strategy quickly becomes unsustainable for a highly connected home - especially when data analytics are desirable to enhance the intelligence and utility of an IoT system. Therefore the free and open-source MySQL relational database is used for data storage which is supported natively by NodeRED.

MySQL is capable of all the functions required for storing and accessing data in the smart home context, while it may not be optimal for time series data, the performance is acceptable at this scale - this decision is justified by the fact that a single storage technology simplifies management and deployment. However for more demanding users there is no reason that a different technology could not be used in harmony with the rest of the system, due to the decoupled nature of flow programming.

4 Demonstration System

4.1 System Operation

Flow based programming lends itself naturally to event based behaviour, and the system generally works on this basis. Its operations can be divided into four

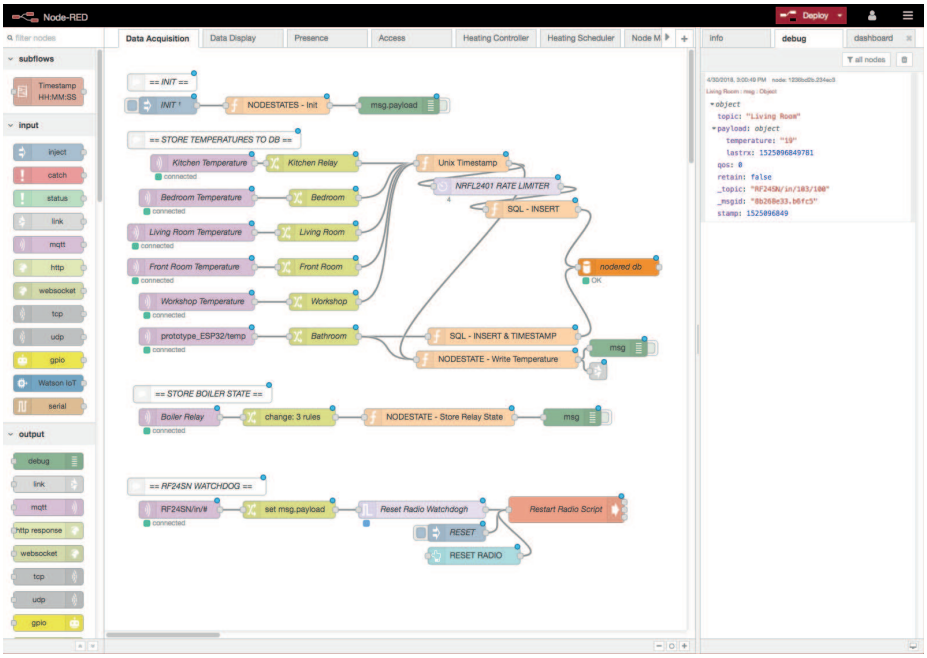

Fig. 2. Flow based programming in Node-RED

categories: Data Acquisition & Storage, Data Processing, System Actions and Interface. Persistent variables are used to define various states of the components and are referred to for the purposes of control and behaviour.

The flow of activity through the program follows the paradigm of information input (be it through sensors, pre-programmed triggers or human action), knowledge extraction by analysis of this information, action upon this knowledge and, finally, display of the relevant extracted data and system actions in the user interface. As the program runs concurrently, all parts of this sequence are simultaneously active - however a change or new event could be observed to ripple through the system in this manner.

Data Acquisition & Storage - Data from the nodes is reported at regular intervals via MQTT messages. Upon receipt in Node-RED, the message is timestamped and its topic set to an appropriate label. If the data is a sensor reading then a MySQL query is generated and the information is inserted into the appropriate database. A NODESTATES object tracks the status of all nodes in the system (including battery voltage, last transmission and online/offline status), and is updated upon receipt of information from each component.

Instantaneous or otherwise non-persistent data can be delivered directly to output nodes (charts, gauges, text) and will not bloat the database with unde-

sired information. The low reliability of the NRFL2401+ radios necessitates multiple transmission attempts by the nodes and this can result in duplicate MQTT messages being received (and therefore duplicate database entries) - however amongst its various filtering functions, Node-RED incorporates a rate limiter that can effectively discard the duplicates based upon the sampling period. Embedding this approach to scoped data display and logging helps to address the issue of surplus, redundant data that plagues IoT.

Basic presence detection is also implemented by setting static IPs for the smartphones belonging to each occupant of the home, the system then pings these IPs at a regular interval to determine if the house is occupied. Whilst there are many options for making this determination (such as GPS based technology), the advantage of this method is that no additional equipment is required and no apps or services are necessary on the smartphones.

Data Processing - As time series data is available in the database for every temperature sensor in the house, many forms of analysis can be conducted. This demonstration computes a windowed derivative to estimate the rate at which each room is heating or cooling, averaging the temperature readings over two periods - the width of the period and the interval between them is customisable. MySQL queries based upon the specified periods are generated and the returned results are analysed with a function written in javascript.

The gathered temperature data will contain noise as well as periods where consecutive readings are the same which is unhelpful when displaying the data graphically. Hysteresis based reporting with a customisable threshold has been implemented for smoothing the raw information which is pulled from the database for display purposes - this produces aesthetically pleasing graphs that clearly show trends whilst preserving the original full data set.

These two data processing activities are simple examples and it should be noted that Node-RED can integrate with external functions (making system calls and running scripts) and also that the data itself is stored in an external MySQL database - advanced analytics can be performed and their results returned, there is no requirement to implement all functionality within the confines of javascript.

System Actions -

Error Identification:
As an IoT system is based upon the idea of networked nodes, it is important to be aware of the health of the components. This system interrogates the NODESTATES object on a regular basis to check for radio errors, low voltage and offline state. A flag based system will indicate if any errors are present and this generates a human readable string to alert the user that some action must be taken. The central heating system is also flagged if the state of the control relays do not match the system demand.

Heating Controller:
This demonstration system has been installed in an old property which previously had a simple timer controlled heating system, the upgraded framework offers smart thermostatic control and remote management. The two rooms in which most time is spent are chosen as the targets for the heating controller, which will base its actions upon the room which is coldest. This multi-thermostat behaviour ensures that a comfortable temperature is achieved in all the living areas - alternative approaches can be applied to reduce energy usage such as ensuring at least one room is comfortable temperature or working on the average of the household.

The boiler is controlled using traditional hysteresis behaviour ie. heating to slightly above the setpoint and then cooling down to slightly below it, and has customisable hysteresis width in each direction. The setpoint can be defined by choosing one of several presets or adjusted in 0.5°C increments, and also by scheduled activity within Node-RED. The target temperature is reduced by a customisable amount when the house is unoccupied and the whole system can be disabled if desired via a GUI switch.

Access Control:
The access control section maintains a list of IDs and their privileges, as well as the logic for non-persistent display of events, battery voltage and last access. See *Nodes:Actors* for more information.

Interface - The lowest bar for IoT in the home is that it is not more difficult than the traditional technology that it replaces. Where it is advantageous, individual nodes should provide feedback without the need to access anything on a tablet, smartphone or computer - for example, in this system the heating controller activates LEDs when the relays are on and the access control node incorporates a piezo buzzer to inform the user of the lock state.

With that stipulation being met, an IoT system should then go on to offer enhanced information, insightful knowledge and new functionality to the user. Node-RED provides an excellent set of user interface and data display widgets that can be used to quickly build an interface that is suitable for all major mobile and desktop platforms - without the need to write any code.

The demonstration system is organised into tabs, two of which are shown in Figure 3, the default display which details overall status, 2 day historical temperature and live readings, and Figure 4 which offers a management interface for the heating system and communications. There are additional tabs showing 7 day historical temperature, access logs and detailed node status.

These tabs, and their contents can easily be re-arranged and customised as different functionality is developed in the system. Adjustments of this kind require no reboot or reset and the tools provided by the Node-RED dashboard are sufficiently feature rich to provide all of the functionality that most home IoT implementations would require.

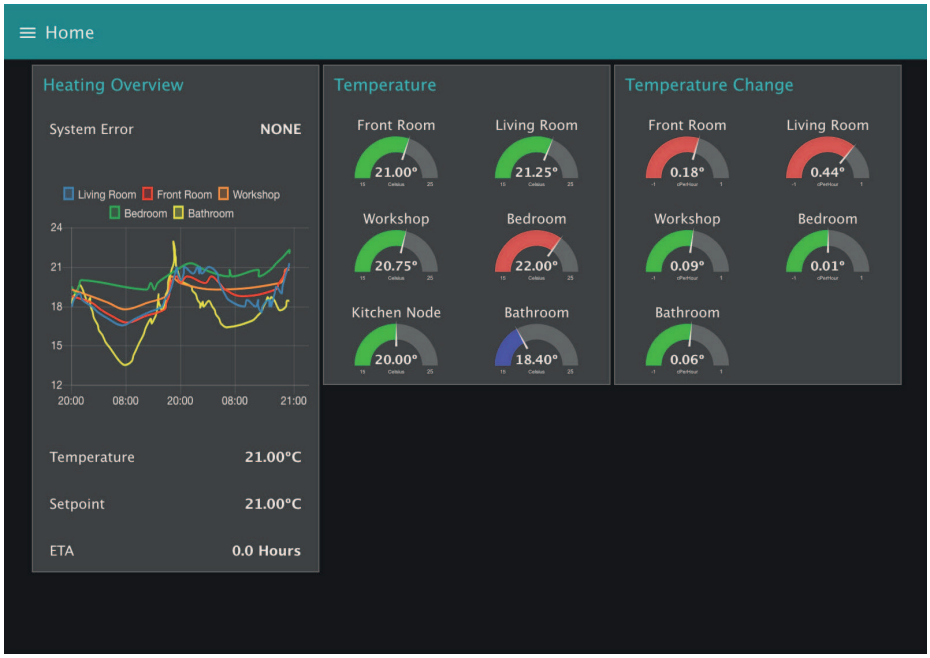

Fig. 3. Web Interface - Home Tab

4.2 Hub

A Raspberry Pi 3 Model B serves as the hub for the system, integrating WiFi with an additional NRFL4201+ radio for wireless communication. It requires hard-wired power and careful location due to the limited performance of the NRFL4201+ radio. The collected tools can be distributed as a preconfigured Raspberry Pi disk image that provides everything required for the IoT hub with a minimum of setup required.

The hub runs the RF24SN service that is responsible for translation between the nRF24L01 radio and the MQTT broker and which allows a single float number to be sent or received per MQTT message, without requiring a full MQTT implementation on the node, or any special adaptations in Node-RED.

4.3 Nodes

The node types can be divided into three groups - sensors (regularly report some feature of the environment), actuators (listen for commands from the hub and then take some action) and actors (perform some action in response to an event but are not available for command unless externally triggered). The following sections gives an example of each, and their place within the system. This grouping should not imply that each node performs significantly different processes,

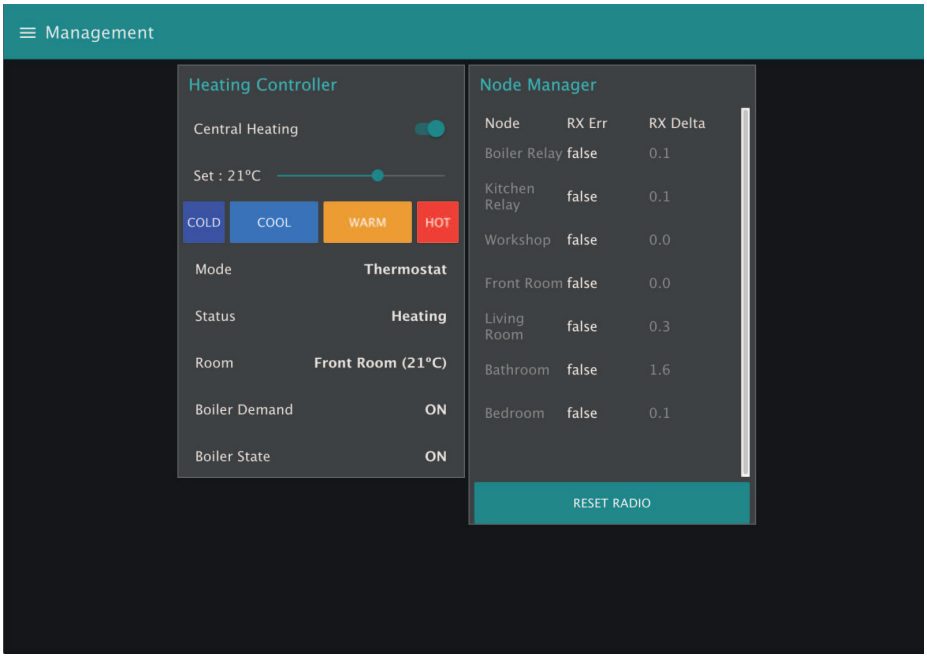

Fig. 4. Web Interface - Management Tab

it is rather the order of operations and their availability to the hub that defines their classification. This generalised approach to creating node functionality that can lend itself to a wide range of applications while maintaining a familiar format for code to maximise reusability.

There is a mix of battery powered and mains powered nodes - any battery powered nodes use their onboard ADC to read the battery voltage and report back to the hub each time they communicate - they will also automatically power down and issue an alert if the charge falls below a programmed threshold, to avoid damaging the lithium cells. No tight tolerance components are required as each node can be calibrated in software for sufficiently accurate readings.

Definition of Operations:
MEASURE - Measure some variable, eg. temperature, battery voltage.
ACT - Take some action eg. operate a relay.
REPORT - Report some information to the hub eg. sensor reading, node state, details of an event.
LISTEN - Poll the hub for instructions and act upon any valid reply received, otherwise take no action eg. OTA Update, Shutdown, Restart
SLEEP - Go to an idle or low power mode for a programmable period of time or until externally interrupted. In the case of battery powered nodes they will transition to the deepest sleep state available, whilst mains powered nodes will

suspend program execution temporarily.

Sensors
Order of Operations: MEASURE - REPORT - LISTEN - SLEEP
Sensors are configured to take readings of some environmental variable, and are expected to be available to the hub on a regular basis. They can implement hysteresis based reporting to save battery but must not exceed a set maximum time between radio contacts, even when there is no change or event to report.

Example : Room Temperature Sensor
The node is composed of an Arduino Nano (a low cost ATMEL based, breadboard compatible microcontroller that has significant support in the form of an active community and many open source libraries) with a NRF24L01+ radio, to demonstrate the possibility of interconnecting diverse technologies to a single hub.

Temperature is measured using the DS18B20 digital sensor, which uses a One-Wire communication interface and offers ±0.5°C accuracy from -10°C to 85°C. This sensor is popular amongst the open-source community and preconfigured libraries exist for almost every available IoT platform.

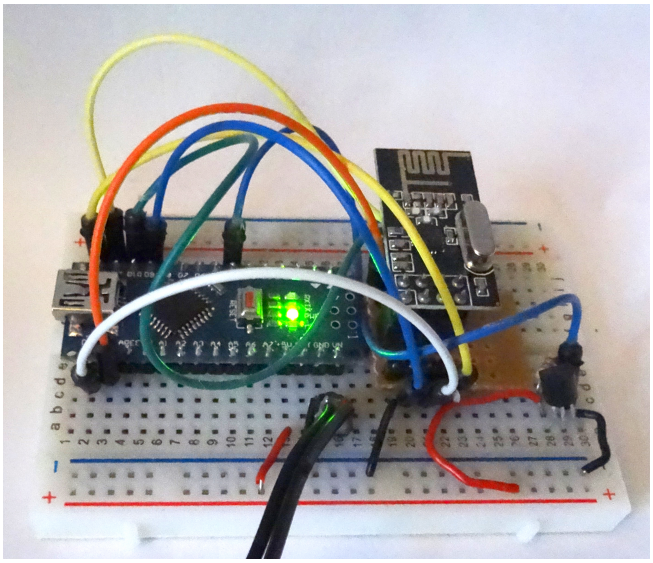

Fig. 5. Arduino Nano based temperature sensor on development board

Actuators
Order of Operations: LISTEN - ACT - REPORT - SLEEP

Actuators are expected to be always online, with minimised latency - and therefore are generally mains powered. The hub will issue commands to these nodes and receive a report of the outcome of the node action. In the case of important systems, the nodes should have suitable fallback/offline behaviour to cater for network failures.

Example: Central Heating Boiler Relays
For most boilers, control is achieved simply by providing or removing mains voltage at their input - with an internal thermostat on the boiler regulating the water temperature. More complex systems that involve ported valves are also actuated by mains power which is switched based upon the controller logic. These systems take discrete states and use the principle of hysteresis for their regulation.

Therefore in the vast majority of cases, the retrofit required for an IoT controlled boiler is simply replacing the original heating controller with relays that are controlled by a networked device. As the device is networked, the demand logic for on/off or any valves does not have to be implemented on the node itself (and can therefore be based upon the rich set of inputs available at the hub), but must incorporate some fallback behaviour in case of network loss.

Upon network loss this node will maintain its state until reset. Upon reset if there are no instructions, the node will turn the boiler relays on - this means that in the worst case, the system can be manually controlled simply by powering the node on or off as desired.

Although the focus here has been centred on the question of controlling a boiler, the implementation of a relay can be carried to many applications in the home - any case in which a person operates a switch can be automated with a relay. WiFi connected relays are widely available from £5 upwards, most of which can be incorporated into this framework.

Actors
Order of Operations: ACT - REPORT - LISTEN - SLEEP
Actors are distinct from actuators because the hub cannot command them directly - there is no requirement for a minimum contact period. They are event triggered devices whose functionality is enhanced by a server connection but not dependent upon it. By definition they should have strong fallback behaviour in the case of a network loss, so that their core functionality is not compromised.

Example: Fingerprint Scanner with Door Latch
A Lolin32 Lite running the open-source MicroPython firmware (which implements a subset of Python 3 for microcontrollers) is interfaced with a low cost fingerprint sensor, the sub-£20 GT511C1R and accompanied by the circuitry required to drive an electronic door latch.

The node is completely battery powered, and spends the majority of its time in a low power consumption deep-sleep mode. When a finger is pressed upon the sensor, this causes a change in capacitance of the surrounding cage - triggering

an interrupt in the ESP32 that wakes the node so that it can scan the fingertip and take appropriate action.

Each saved fingerprint is assigned an ID ranging from 0-19, with unidentified fingerprints reported as -1. If the node reads a fingerprint in the range of 0-9 the door will open immediately and the event is reported to the hub. If any other ID is read, the node will transmit the ID and query the hub for the appropriate action - opening or keeping closed the door as specified, and will default to closed if no answer is received. Note that the hub enhances the functionality of the node, but this critical function can still operate correctly in the case of network failure.

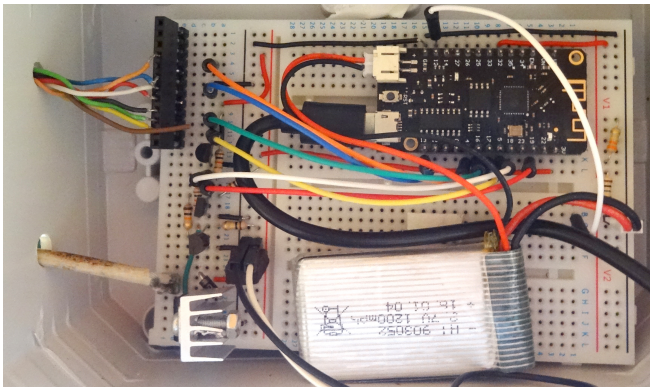

Fig. 6. Lolin32 Lite (ESP32) based node with circuitry for fingerprint scanner and door latch control on development board

5 Discussion

This paper has been focused on the introduction of the system and framework, however it is beneficial to touch upon the subjects of practical implementation and the data gathered to date. The following notes serve to ground the ideas put forward thus far and will be the subject of future development.

The low cost 2.4GHz radio system from Nordic Semiconductor does not support WiFi or encryption and its use here should not be considered an endorsement of this outdated technology which has largely been supplanted by low cost devices integrating 802.11 WiFi. Espressif has made a large impact with their low-cost WiFi microcontrollers, firstly with the ESP8266, followed by the upgraded ESP32. The SoCs can be procured for less than £2 and development boards for less than £5 - this price point made the aforementioned NRF24L01+ practically obsolete for IoT applications and due to the fact that it must be paired with a microcontroller, the total cost is usually higher than a single ESP SoC.

Battery powered nodes have notoriously tight power budgets and the Lolin32 Lite is no exception in this case. The specification of the ESP32 claims a deep sleep power of 5.5μA, whilst the Lolin32 Lite has been measured at 1.4mA which, unfortunately, is considerably too high for a practical sensor. Some preliminary analysis points towards low quality support components on the PCB and will necessitate the development of a custom carrier PCB with much lower passive power usage. Continuing hardware development of the system will concentrate upon achieving >1 month battery life, which can be considered a practical minimum for a multi-node installation.

Examining the collected time series of temperatures (along with knowledge of the house activities) has made it clear that rich knowledge is available from this data - it is possible to easily see when the shower has been used, to determine if doors between rooms are open or closed, the thermal characteristics and performance of individual rooms are readily apparent and faults in the heating system leave unique features. The potential for intelligent behaviour when more advanced analytics are applied is quite significant - additional sensors and measurands will only increase this. Minimising the fuel used for heating through smarter scheduling and recommendations for targeted insulation efforts (identified by modelling the performance of the various house areas) could make a significant impact on energy bills.

6 Conclusion

A framework based upon flexibility, wide compatibility and quick reconfiguration was proposed for Home IoT applications. Using open-source software throughout reduces cost and increases transparency of each component, safeguarding personal data.

Due to the changeable and unpredictable nature of IoT, the framework does not seek to impose limitations or anticipate end uses - instead it offers a generalised set of tools to easily accomplish tasks and respond to changing application requirements.

A demonstration system which has improved the performance of the home in which it was installed has been described, and some basic real-time analysis of the data has been incorporated without the need for any third party tools. Informal observation of the data suggests that there is significant scope for knowledge extraction from low-grade sensor information and with more advanced analytics it would be possible to make suggestions for physical improvements that target heating efficiency, as well as automatically minimising the fuel required.

References

1. D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, 2011.

2. S. Zitnik, M. Jankovic, K. Petrovcic, and M. Bajec, "Architecture of standard-based, interoperable and extensible IoT platform," *24th Telecommunications Forum, TELFOR 2016*, pp. 0–3, 2017.
3. C. H. Lu, "Improving system extensibility via an IoT-interoperable platform for dynamic smart homes," *Proceedings of the 2017 IEEE International Conference on Applied System Innovation: Applied System Innovation for Modern Technology, ICASI 2017*, pp. 1300–1303, 2017.
4. S. Dharur, C. Hota, and K. Swaminathan, "Energy efficient IoT framework for Smart Buildings," pp. 793–800, 2017.
5. S. Leminen, M. Westerlund, M. Rajahonka, and R. Siuruainen, "Towards IOT Ecosystems and Business Models," 2012, pp. 15–26. [Online]. Available: http://link.springer.com/10.1007/978-3-642-32686-8_2
6. M. C. Ruiz, E. Mcarmenruizuclmes, T. Olivares, E. Teresaolivaresuclmes, and J. Lopez, "Evaluation of Cloud Platforms for Managing IoT Devices ."
7. A. Connolly, "Freedom of Encryption," *Real-World Crypto*, no. February, pp. 102–103, 2018.
8. N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*, 2017.
9. A. Belsa, D. Sarabia-jácome, and M. Esteve, "Flow-based programming interoperability solution for IoT Platform Applications," 2018.
10. A. Rajalakshmi and H. Shahnasser, "Internet of Things using Node-Red and alexa," *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*, pp. 1–4, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8261194/