LEVERAGING INTERNET OF THINGS DEVELOPMENTS FOR RAPID PROTOTYPING OF SYNOPTIC DISPLAYS

L. T. Stant*, T. Cobb, Diamond Light Source, Oxfordshire, UK

Abstract

Recently the technology industry has been laying foundations for the eponymous Internet of Things (IoT): efficient publish-subscribe protocols; process control schemas for household items; and improved low-power radio communications. Accelerator controls and IoT have several aspects in common - small payloads, low latency, dashboard/synoptic data presentation format are some examples. The IoT now provides several open-source projects which can provide a partial implementation of one or more accelerator controls software features. Because development is typically a lower priority for accelerator controls groups, there is a valid case to try and utilise the free efforts of others for the benefit of accelerator controls. In this paper, the authors present examples of the use of IoT frameworks for synoptic display/GUI development. The work in this paper enables other developers to access this resource and experiment with their own systems.

INTRODUCTION

The IoT is a long-term upgrade of our global interconnectivity which aims to increase the efficiency with which we go about both our personal and professional lives. It will achieve this by adding remote interfaces to everyday items, deferring tedious interactions to increasingly intelligent computer algorithms. The deployment of this technology is slower than expected, especially as it requires standardisation and harmonisation which is less favourable when vendors are developing the technology.

However, we are starting to see the IoT enter our homes in the form of lamps and thermostats. In addition to commercial products, various suppliers along with the open-source community are now offering development platforms and software implementations to facilitate custom IoT solutions. Two of these solutions are of particular interest for controls applications at accelerators - "dashboards" and portable intelligent displays - which we have started to research for use with the EPICS deployment at Diamond Light Source (DLS).

DASHBOARDS

The IoT uses software dashboards as a synoptic user interface (UI) to allow a broad overview of sensory data acquired from distributed devices. An example application could be location trackers attached to livestock, with the dashboard showing time spent at various locations and alarm notifications for individuals moving outside of the allowed area. This UI has a clear analogue with accelerator controls: the livestock are items of equipment, the UI shows performance and status data, and alarms check parameters against operating limits.

Where IoT dashboards can differ from classical controls synoptic UIs is extensibility and visual design. Dashboards need to be created by the user, so UI items are modular and easy to reconfigure. Most dashboard projects solely focus on the UI software itself, so more time is devoted to making the UI visually appealing and attractive. Although for many controls applications the latter point is not important, sleek synoptic displays are often desired around control rooms and high-level status screens. Another benefit of IoT dashboards is that they are often built to be fully responsive to different screen sizes and aspect ratios, including mobile devices. Existing UI solutions, such as Control System Studio [1], offer scaling of the display but are still not optimised for large interfaces on small screens. Where IoT dashboards are typically hosted as a web page, this means that a single interface can be designed and easily accessed on both workstations and mobile devices.

Node-RED Dashboard

Node-RED [2] is a locally hosted web-based visual programming tool for the IoT, written in Node.js [3] (a Javascript framework). It is similar in function to other websites which aggregate and translate packets from different IoT standards (e.g. IFTTT [4]), except that Node-RED uses flow-based graphical programming for configuration. Included in the base software is a UI dashboard, which communicates with the backend via WebSockets (see Fig. 1). The project is open source (Apache 2.0), and it is possible to implement new UI components. The responsive grid layout is well implemented and orders can be assigned to groups of components to help with arrangement on small displays.

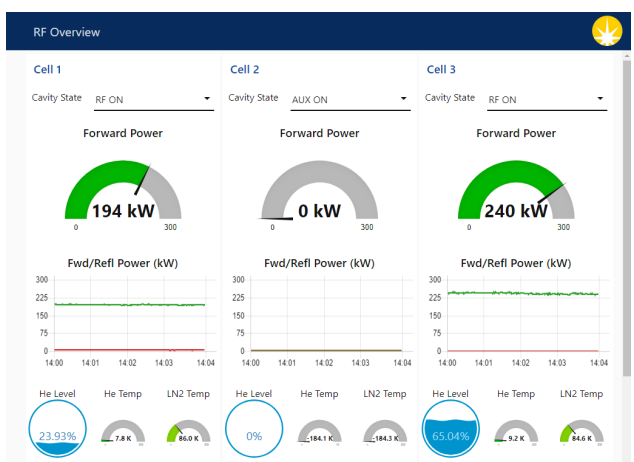

Figure 1: Example Node-RED UI dashboard showing a simple synoptic display.

* laurence.stant@diamond.ac.uk

Grafana

Although designed for use with monitoring IT infrastructure, Grafana [5] is becoming popular with IoT users due to it's sleek appearance and built-in alarm features. Instead of subscribing to updates as is the case for typical IoT dashboards, Grafana data source providers connect to time-series databases and retrieve archived data. This is similar to existing EPICS tools such as Strip Tool, and therefore it can be used as an alternative. Features which are particularly useful for accelerator controls applications are the alarm handlers and post-processing access. Alarm limits can be set on the dashboard, triggering customisable notifications to a variety of endpoints, and occurrences can be logged to an annotations database for reporting and labelling plots in future. An active alarm UI component can be included to tabulate existing alarms, although they are also displayed on the source component (e.g. graph, gauge). Grafana also provides convenient configuration of the data source "metric" for each UI component, whereby a user can select maths functions to apply to the data for averaging and other functions. Like Node-RED, Grafana is also open source (Apache 2.0), locally hosted and has a responsive layout. An example synoptic screen can be seen in Fig. 2.

Figure 2: An example synoptic UI made with Grafana.

PORTABLE DISPLAYS

Monitoring of process variables (PVs) normally requires a PC workstation to be installed at the location. This poses no issue when many PVs are to be displayed as a screen, or if the workstation is located in a control room or cabin. However, consider the following use cases:

1. Local readout of parameters for headless equipment, both temporary (user) or permanent. This could include both information from the equipment (e.g. position information from stages) and that from the control system, such as operating modes and sequence information.
2. Desktop monitors to allow constant access to crucial values while PC screens may be used for other work, or reduce workspace changes. Rather than repeatedly hunt for a value on a crowded display, a dependent PV could be sent to a dedicated display mimicking a physical readout.
3. Portable maintenance readouts. These devices, preferably wireless, are configured with one or few temporary PVs and can be magnetically attached to equipment such as cabinets and pipework while maintenance is performed. The technician has a real-time monitor of the values read by the machine.

For these use cases, a device smaller than a typical PC workstation is required. We will now look at some options, including new possibilities from IoT developments.

Mobile Phone

Mobile phones capable of internet connectivity have been available for many years, providing controls developers with the possibility to make custom web-based interfaces for portable or desktop monitoring (and control) of PVs. Instead of writing a new interface, IoT developments now provide many dashboard "apps" [6–8] which operate in a similar way to those described in the previous section, although truly optimised for mobile phone displays. These apps connect to IoT protocols, so middleware is required to connect to control system PVs. This will be covered in a later section.

Raspberry Pi

The Raspberry Pi single board computer [9] is a widely used platform for education and development. It is also very convenient to use wherever a linux-capable, small form-factor networked device is required. As such, it fits the use cases listed above well. Many add-on hardware "shields" are available, including displays with button interfaces as shown in Fig. 3. Newer models are Wi-Fi capable, while others can be made so with a usb device not much larger than the port itself. Although there are IoT monitoring software implementations available online [10], the capability of running Linux means the Raspberry Pi can use a standard channel access client without the need for middleware.

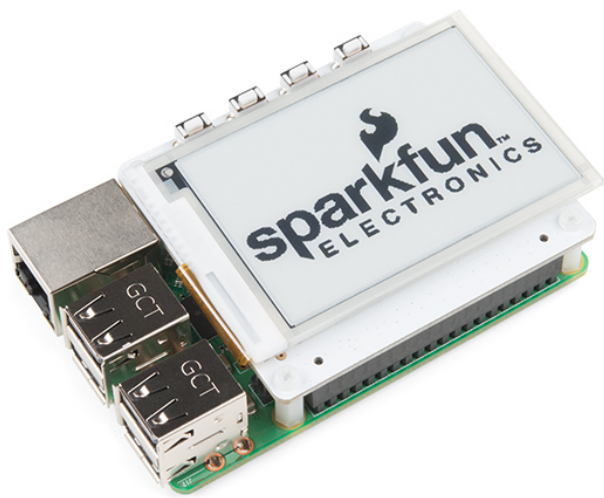

Figure 3: Raspberry Pi single board computer with plugin module featuring an epaper display and tactile button interface.

ESP8266 Platforms

The ESP8266 is a 16-bit microcontroller with built-in Wi-Fi capability, which is either available as a single 5x5 mm QFN32 chip, or, with memory and antenna as a 14x25 mm module. The low price point and ability to deploy code from the popular Arduino IDE has led to it becoming the most popular device for open-source IoT device implementations. The module is often paired with some useful I/O - typically a display, sensor interface, or relay module for driving outputs. It also consumes much less power than Raspberry Pi by using a deep sleep mode, which makes small battery power sources practical (100-300 mAh). Many IoT hardware designs using this module also include a lithium polymer battery charging circuit. There are many open-source implementations of IoT monitoring code for this device, written for a variety of display types and sizes [11–13]. Three popular choices of display are TFT, OLED and ePaper, where the lower power consumption of the latter is advantageous.

ESP8266-based displays appear to be a good solution for portable wireless PV monitors, but where can they be reliably supplied? There are several options available:

1. Develop custom hardware in-house. This is accessible to facilities with standard printed circuit board (PCB) design capability, as there is no specialist ability needed for routing these components. The ESP8266 chip has a reliable supply, as does the alternative 32-bit ESP32. The module can also be procured in several formats, but using a second PCB will increase the size of the finished unit.

2. Procure development boards from online import retailers. There are several well-known importers of electronic goods which stock variations of the same configuration (e.g. ESP8266 and OLED screen with LiPo charger) from many different manufacturers. Although one manufacturer may not be reliable, the low price means that boards can be procured from several to solve any issues. These boards do not include batteries or cases, hence any parts required for a finished device will need to be acquired and assembled separately.

3. Procure complete units. Recently, the same online importers have started to stock the above units complete with cases and batteries. There are fewer manufacturers producing these at the moment, so the supply will be less reliable.

In addition to the above options, 4D Systems® provides an extremely compact device, the IoD-09 [14]. This module, shown in Fig. 4, is only 32x16 mm in size and available from a popular online supplier [15]. Similar to option 2 above there is no battery, charging circuit or case. A 150 mAh battery will fit in the space behind, and a full device with usb charging and magnetic case is being built at DLS.

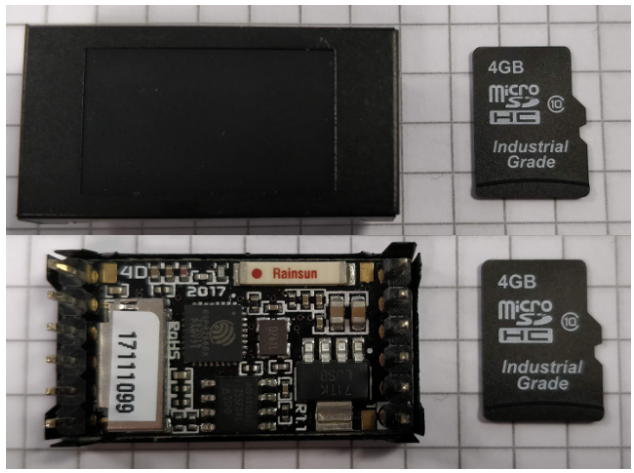

Figure 4: Miniature IoD-09 ESP8266-based TFT display shown alongside MicroSD cards for scale. The square grid has 5 mm spacing.

ARCHITECTURE

Apart from the Raspberry Pi, all of the IoT solutions presented in this paper require either middleware, or plugins to existing software, for them to communicate with the accelerator control system. This section explores these implementations, which are shown connected together in Fig. 5.

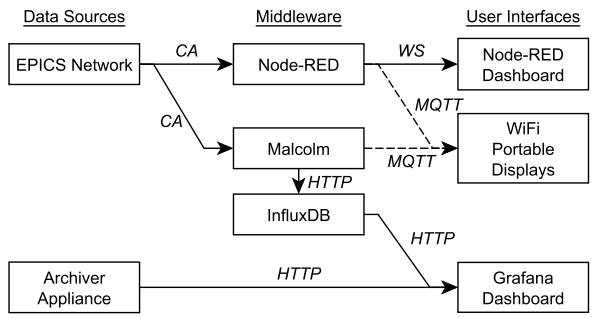

Figure 5: Diagram showing the connections between different components presented in this paper.

MQTT

Message Queue Telemetry Transport (MQTT) is an extremely lightweight broker-based publish-subscribe messaging protocol for the IoT [16]. It is capable of handling large amounts of transactions with reliable transmission via TCP. Because of this, it has been used in applications such as mobile chat [18] and modern railway signalling control [17]. MQTT also includes a quality of service parameter which controls the degree of delivery confirmation.

Client libraries exist for many popular programming languages, and there are several well-established open-source brokers which are simple to deploy and include optional authentication [19, 20]. In addition to PC usage, there are libraries available for IoT hardware devices such as the ESP8266 platform [21, 22]. The combination of a proven

track record and widespread support makes MQTT a great option for implementing IoT devices for accelerator controls.

Node-RED

A natural middleware for the IoT, Node-RED's intuitive visual programming style has made it very popular with users. Nodes are dropped onto a "flow" and connected via wires to configure dataflow. Being written in Node.js, it is optimised for event-driven processing which suits message-based IoT communications. To support new devices and web services, Node-RED allow users to easily define new nodes and host them on a public online repository.

For this work, an EPICS client node has been developed which uses an open-source Node.js channel access library [23] which references libca. The EPICS input node attaches a camonitor to the PV specified in the configuration and emits a new message upon update. This is then routed to an MQTT output node which sends a new value to the broker, which in turn publishes an update to all subscribers. Processing nodes are available for filtering and maths functions to be applied.

Malcolm

Another middleware option available at DLS is Malcolm [24]. This software, written in Python, is designed to perform detector scanning in the middle layer and present a standardised virtual device to data-collection software. This is achieved by providing a modular framework which includes channel access and HTTP plugin "blocks". Blocks can be connected like nodes in Node-RED, although the configuration is read from a text file.

A new block for MQTT was implemented using the paho-mqtt library [25]. Supplementary features such as alarms are not currently supported in Malcolm, but this is planned for a future release.

Initial Grafana support was added by pushing PV updates to an InfluxDB instance using a modified HTTP block in Malcolm. InfluxDB is natively supported as a data source by Grafana and can receive updates via HTTP POST requests.

Grafana EPICS Archiver Appliance Data Source

To avoid creating an unmanaged copy of PV archive data using InfluxDB, a new data source was written for Grafana in Typescript (a superset of Javascript). This provides an interface to the time-series database hosted by the EPICS Archiver Appliance [26]. Whenever a screen refresh is triggered in Grafana (either periodically or by changing the time range), the new time range and required metrics are sent to the database for retrieval. The data source translates between the format of the Grafana and Archiver Appliance requests and responses.

Maths functions, allocated to UI components when configured, must be supported by the data source. The Archiver Appliance includes a range of post-processing maths functions which are included with the data source developed in this work.

FUTURE WORK

Management Interface

Configuration of the portable displays is currently achieved by reprogramming the device. This task can be made far easier by providing a web UI which pushes new configurations via MQTT. The device will update immediately to reflect the changes. Users can select the number of PVs and include multiple pages if required, specifying an optional delay for cycling between them in addition to button input.

LPWAN Communications

The IoT is also innovating communication at the hardware layer, driving several new standards in Low Power Wide-Area Networks (LPWANs). These networks provide reliable lightweight communication typically over 5 km in urban environments, and are designed for low power sensing nodes. Although this paper has only studied IoT developments for monitoring purposes, LPWAN technology could provide convenient and reliable wireless inputs to control systems as an alternative to Wi-Fi.

Wearable Devices

Wearable technology is enjoying huge popularity at the moment, with many consumer electronics companies providing new models frequently. These devices, specifically "smart watches", offer a high quality version of a portable display but suffer two disadvantages when compared with those mentioned earlier - they are significantly more expensive and do not offer a consistent development environment for new applications, if at all. However, cheaper versions of these watches are becoming available of which many use the same Wear OS operating system [27]. Some devices do not support Wi-Fi, instead connecting via Bluetooth to a nearby mobile phone. For these devices to connect to EPICS PVs, a service would need to be running on the mobile device.

CONCLUSION

This paper has presented a summary of recent IoT developments which can be utilised by the accelerator controls community to implement synoptic displays and portable PV monitoring devices. We have shown that it is possible to deploy a compatible IoT system with little effort, due to the similarities of the communication requirements between the two areas. Input to the control system has not been addressed due to the lack of maturity of this technology for the application, but may be investigated in the future.

REFERENCES

[1] Control System Studio home page, http://css.desy.de/content/index_eng.html
[2] Node-RED home page, https://nodered.org/
[3] NodeJS home page, https://nodejs.org/en/
[4] IFTTT home page, https://ifttt.com/
[5] Grafana home page, https://grafana.com/

[6] IoT MQTT Panel app home page, https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod

[7] Blynk app home page, https://play.google.com/store/apps/details?id=cc.blynk

[8] Cayenne app home page, https://itunes.apple.com/gb/app/cayenne-iot-project-builder/id1057997711?mt=8

[9] Raspberry Pi home page, https://www.raspberrypi.org/

[10] PaPiRus-MQTT Github page, https://github.com/Dom1n1c/PaPiRus-MQTT

[11] esp_mqtt_oled Github page, https://github.com/nathanchantrell/esp_mqtt_oled

[12] e-ink-display-esp8266-mqtt-openwhisk Github project, https://github.com/timwaizenegger/e-ink-display-esp8266-mqtt-openwhisk

[13] AZSMZ-EPAPER Github project, https://github.com/cxandy/AZSMZ-EPAPER

[14] IoD-09 product page, https://www.4dsystems.com.au/product/IoD-09

[15] RS home page, http://rswww.com

[16] MQTT home page, http://mqtt.org/

[17] L. Zhang, "Building facebook messenger", Aug. 2011, https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920

[18] D. Wood and D. Robson, "Message broker technology for flexible signalling control", Mar. 2014, http://www.irse.org/knowledge/publicdocuments/3.09%20Wood%20-%20Message%20broker%20technology%20for%20flexible%20signalling%20control.pdf

[19] Eclipse Mosquitto MQTT broker home page, https://mosquitto.org/

[20] EMQ MQTT broker home page, http://emqtt.io/

[21] pubsubclient Github page, https://github.com/knolleary/pubsubclient

[22] esp_mqtt Github page, https://github.com/tuanpmt/esp_mqtt

[23] node-epics Github page, https://github.com/RobbieClarken/node-epics

[24] Malcolm home page, https://pypi.org/project/malcolm/

[25] paho-mqtt home page, https://pypi.org/project/paho-mqtt/

[26] M. Shankar, M. Davidsaver, and M. Konrad, "The EPICS archiver applicance", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 761-764. doi:10.18429/JACoW-ICALEPCS2015-WEPGF030

[27] Wear OS home page, https://wearos.google.com/