Offline Microcontroller Network Based Monitoring System with a Configurable Sensor Set

Roland Hobaj
Codespring
Cluj-Napoca, Romania
hobaj.roland@codespring.ro

Csaba Sulyok
Babeş-Bolyai University
Cluj-Napoca, Romania
csaba.sulyok@gmail.com

***Abstract*—The accessibility of information is a key factor in the progress of modern businesses. The Internet of Things technological paradigm presents new possibilities and challenges with regards to optimization and monitoring of production workflows. The current paper presents a configurable monitoring system for automation of startups and agricultural companies. The software provides a management/monitoring solution for enterprises running in lack of Internet connections; it allows the attachment of microcontrollers with configurable sensor sets. The project also contains a mobile client aimed at wireless system configuration and monitoring. We demonstrate the application by tackling real-world problems from the apiary industry.**

I. Introduction

Smart devices and equipment influence many areas of modern day living [1]. The fundamental advantage of Internet of Things (IoT) systems is the ability of the many small components to interact with each other via a local or global Internet connection. This gives the opportunity to measure, process and optimize environmental indices. With this technological paradigm, enterprises manage to redesign business processes and factory workflows [2], [3]. IoT systems therefore gain noticeable attention and acclaim from a wide range of industries.

A common design standard for microcontrollers is accomplishing one exact assignment, in contrast to microprocessors with general purpose usability. Ordinarily, microcontroller units (MCU) gather information via a specific, rigid sensor set. Data collected from the measurement instruments is processed and forwarded to other components of the overarching system. Microcontrollers must serve real-time feedback to events in case of predefined rule violations.

Taking the cost of the enterprise software design and development into consideration, a full automation for startups or young firms may prove a cumbersome investment. In case of agricultural companies, the usual location of target systems–remote fields–also presents a drawback: the lack of a stable Internet connection. A potential solution for those businesses can be an offline monitoring system with reusable MCUs and configurable sensor sets.

This paper aims to demonstrate the implementation of the SenseIT project[1]. A React Native [4] mobile client provides the opportunity for users to browse and configure available microcontrollers in the local network. Online MCUs are discovered by a central master Node.js [5] server, which manages a user-defined sensor rule list. Data gathering, real-time events and their handling is made possible by the connection of an arbitrary number of either Particle Photon [6] or Raspberry Pi 3 [7] devices.

[1]Project hosted open-source at https://gitlab.com/senseit

II. Motivation

This section presents the central motivation of the project. The main aspects may be summarized in the following three topics.

A. *Digital business*

The Internet of Things is widely praised as the fundamental technological paradigm of the imminent Fourth Industrial Revolution [2], [3] (Industry 4.0). This innovation generates the transformation of modern living systems like business, health care, education, etc. Gartner [8], an acknowledged management consulting firm, estimates about 20 billion devices connected to the Internet by the year 2020, with the percentage of digital businesses related to traditional enterprises rising to 65%. In the Industry 4.0, the accessibility of information and knowledge is a demanding aspect of success for modern companies [2]. The diversity of IoT system applications and a wide collection of smart devices can make it challenging to identify the business values [8].

The information gathered by microcontrollers and sensors is processed by a software system. The cost of software development for personalized aims can present a barrier for startups and young firms in becoming digital businesses. The SenseIT project is a general-purpose IoT software in which clients can configure the system with various number of Particle Photons and Raspberry Pis equipped with dynamic sensor sets. Thus the entrepreneur manages to avoid the expenditure of software development during the automation of the production workflow. There is no single industry branch for MCU maintenance; we instead propose a generalized approach, tested and exemplified through apiary use cases.

B. *Exchangeable sensor sets and MCUs*

Through the growing process of a startup, the monitoring and information gathering requirements are volatile. A

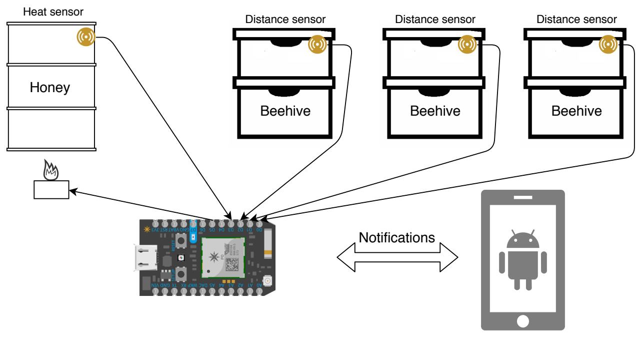


Fig. 1. Beehives with distance sensors

potential solution for an apiary with a modest number of beehives would be the software system equipped with distance sensors wired to a Particle Photon microcontroller (see Figure 1).

Beehives are observed by sensors, notifying the apiarist when one becomes unsealed. A recurring task in apiary management is bee treatment with a special smoke against the external parasite Varroa destructor [9]. This procedure is performed twice a year: once in the spring and again during the autumn. The application of smoke has to be repeated after two weeks during the treatment process. The software system gives the opportunity for users to set up triggers and create reminders. When the apiarist opens up the beehive for the first operation, he/she gets a notification on their smartphone, and the application provides the possibility to set a reminder to repeat the treatment. The created reminder event appears on the native calendar of the phone.

Another possible use case addresses the long term storage possibility of honey, due to its chemical properties. Honey crystallizes after a given time depending on its type; the melting point of its crystallized form is approximately 40° C [10]. Overheating degrades the quality and destroys beneficial enzymes, or caramelizes the sugar. In order to pack honey for sale, it undergoes a carefully controlled heating process. To prevent overheating, the system can turn off the electrical heat source when the temperature reaches a threshold. With a heat sensor in the honey barrel acting as input, and the electrical interrupter connected to the Photon as the output, the system preserves the content at a narrow optimal temperature interval.

The presented scenario and topology provides an intuitive sample for the practical implementation of the project, but it can be configured uniquely based on diverse production workflow requirements. The system may also be scaled up in case of overarching company growth, since the master server is able to serve a large number of microcontrollers.

C. Offline monitoring system

Although project members have to operate in the same local network, the project serves data without an explicit need for an Internet connection. The components of the system

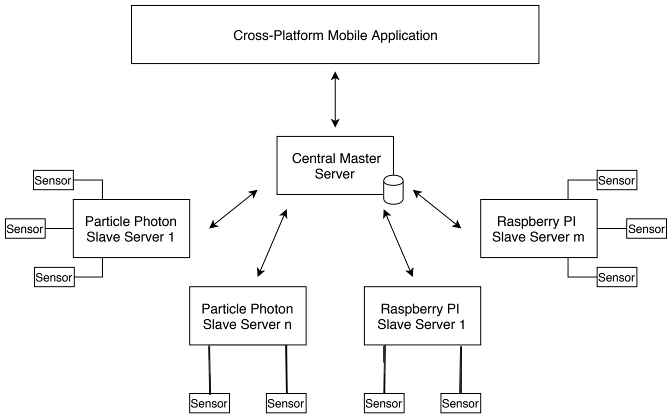

Cross-Platform Mobile Application
Central Master Server
Sensor
Sensor
Particle Photon Slave Server 1
Sensor
Raspberry PI Slave Server m
Sensor
Sensor
Sensor
Particle Photon Slave Server n
Raspberry PI Slave Server 1
Sensor
Sensor
Sensor
Sensor

Fig. 2. SenseIT Project Architecture

communicate with each other wirelessly, except the connection between microcontroller and sensors.

Offline functioning is advantageous in working fields without phone signal or Internet access. In order to gather different types of honey, the beehives are placed on regularly moving trucks, decreasing the feasibility of stable online access even more.

From a security perspective, offline activities are less vulnerable. Besides (or perhaps due to) the increase of the number of online devices, Gartner [8] predict significant increases in the severity and occurrences of Distributed Denial of Service (DDoS) attacks within the next years. The Internet of Things introduces various security challenges [1] by reason of expanding the traditional Internet with ”things” communicating with each other. The sensor data protection is a lower priority task in software security, because any outsider sensors may measure the same values. A more essential issue is the sensor privacy, which describes the events in which the collected data is used.

III. Architecture and Implementation

The SenseIT project is divisible into three isolated components (see Figure 2): the central server, the arbitrary number of microcontrollers equipped with sensors (these may be Particle Photons or Raspberry Pis), and the mobile application. The central server is able to handle more than one microcontroller: the MCUs are shown numbered from 1 to n or m, depending on the microcontroller type.

The inter-component communication uses the HTTP protocol and the endpoint design follows the REST (Representational State Transfer) architectural style [11]. As already mentioned, the master server advertises its presence on the local network, thus the mobile application may discover it automatically. The Particle Photon and the Raspberry Pi deploy an mDNS protocol implementation as well, thus the central server discovers the working slave servers. Although native push notifications would be a plausible solution for presenting changes in the environment, they are served by a third-party server, making an Internet connection

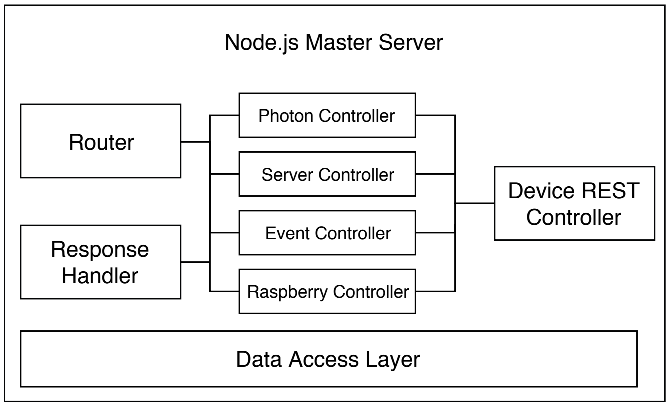


Fig. 3. Node.js master server architecture

indispensable. Therefore the notification system of the SenseIT project is based on the polling mechanism: the mobile client periodically queries the events from the master server, which publishes requests to the online microcontrollers.

A. Central Master Server

The central server is a Node.js lightweight process for administering microcontrollers and sensor rules. The server advertises its presence on the local network via the multicast DNS (mDNS) [12] protocol, thus the mobile application discovers it without user configuration. The main benefit of this protocol is IP address transparency: it discovers names by IP message, sent to the members of the network. Target processes which are subscribed to the mDNS advertising, multicast a message with their own IP address.

The components of the master server application are shown in Figure 3. The *Router* module handles the requests from the mobile clients, and forwards them to the appropriate controller. The *controllers* have permission to manipulate the information in the database through the Data Access Layer. They are also responsible for sharing the user defined information with the MCUs, assisted by the *Device REST Controller*. For instance, when the user uses a mobile phone to create a sensor rule for a Photon, the request is processed and sent by the *Router* to the *Photon Controller*, which stores the data and forwards it to the MCU. The *Device REST Controller* provides a general RESTful API and is used for communication with both types of microcontrollers. This architecture allows easy integration of other smart devices (e.g. Arduino) by the implementation of a new slave server communication through the given API.

The user-defined sensor rules and the microcontroller configurations are persisted in a MongoDB [13], [14] database system. The Mongoose Node.js library provides the possibility of model scheme creation and management with basic CRUD operations. The project uses three essential persistent entities:

- *ServerModel* - server name and port configurations;
- *DeviceModel* - responsible for sensor rule storage;
- *EventModel* - contains the events triggered in the system.

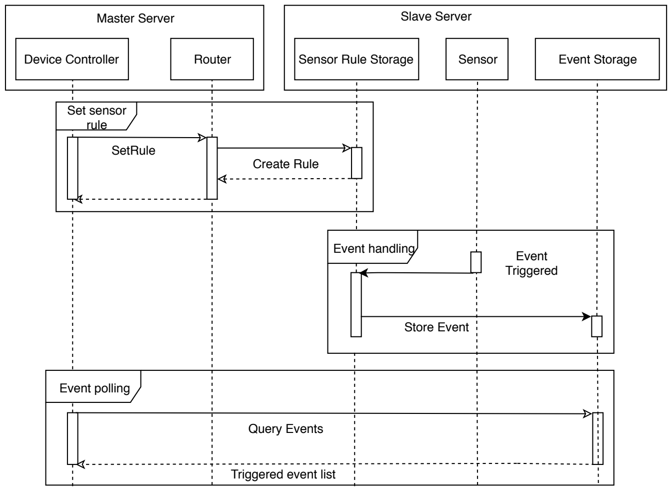


Fig. 4. Event triggering sequence diagram in Photon slave server

B. Particle Photon Slave Server

The Particle [6] is an all-in-one IoT platform from the physical hardware to the cloud. Their devices are provided with built-in connectivity components, such as WiFi chips or cellular modules for communicating the central Particle Cloud.

The Photon device is selected for the SenseIT project mainly because of its integrated Broadcom 802.11b/g/n WiFi chip, essential in our wireless communication system. The modest size of the Photon restricts its resources: it contains a 120 MHz ARM Cortex M3 processor, 1MB flash memory, 128KB RAM, and 18 mixed-signal general-purpose input/output (GPIO) channels.

The *Particle Build* is a browser-based development environment providing sofware development opportunities in a platform-independent manner. It is not a requirement for the microcontroller to be connected to the development device, the new firmware may be flashed to it with the aid of the Particle Cloud. Therefore, the development team and the device are not required to share a physical location.

Accessing the Particle Cloud requires an ongoing Internet connection; considering the motivation of the project, we run the Photons in offline mode. The C++ firmware managing the Photon therefore starts the device in “Manual” mode. This allows the software to connect only to the local network, omitting the cloud.

The central functionality of the C++ Photon slave server is the sensor rule and event management. Figure 4 describes the rule creation and event triggering processes. The Node.js master server forwards rule creation requests from the mobile client, and these are saved into the *Sensor Rule Storage*. This container listens to environmental changes (e.g. a button press, sensor touch, etc.) in the firmware loop function. When an event is triggered, it is saved in the *Event Storage*. The repository allows the central server to query and store the list of occurred events.

C. Raspberry Pi

The Raspberry Pi is a well-known and widespread single-board computer. Equipped with a 1.4GHz 64-bit quad-core processor, the Raspberry Pi 3 B+ chosen for the project is sufficiently robust to run the central master server and an instance of the SenseIT Raspberry Pi slave server as well. Therefore the entire project may be set up without a general-purpose desktop PC or laptop.

The Raspberry SenseIT server has the same functionalities as the presented Particle Photon server. The Node.js process advertises the presence in the local network using the mDNS protocol, whereby the central server is notified that a new device is connected. The user-defined sensor rules are cached in an in-memory database, allowing the central server to handle both devices similarly.

D. Mobile Application

The mobile client is a cross-platform React Native application with Redux as a state management container. React Native is an open source JavaScript library created by Facebook. It allows building native applications using the React library for Android, iOS and the Universal Windows Platform (UWP). The current version of the SenseIT mobile application has been tested exclusively on Android devices.

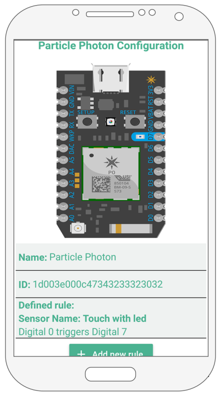
The mobile client is composed of React components, which communicate by *props* in case of parent-child relations, or by *reducers and actions* in case of siblings. The *props* are primitive variables with simple types, e.g. strings, numbers or lambda expressions aiding an asynchronous callback mechanism. Every component has its own state, in which its data is stored, furthermore the application has a general state as well. The latter is managed by the Redux container, and it stores general data, e.g. the IP address of the connected central server. *Reducers* specify the application state change due to *actions* sent to the store.

The history stack creation is realized using the React Navigation library. This community-driven third-party package handles the transitions between screens and offers the ordinary "look and feel" in Android or iOS.

IV. Functionalities

The central component of the mobile application is the event list scroll view (ses Figure 6), which contains the actions triggered in the system. The header allows general navigation options; a user may access the reminders or the server management page. When the application is not connected to any SenseIT server, the scroll view is replaced by an appropriate warning message.

The reminder list component (see Figure 7) presents the saved events and the user-defined tasks. The customer can edit, delete or mark the entries as finished. The application requests authorization from the user to manipulate the native calendar and the created tasks can be stored there with a reminder date. On approach of the reference date, the client receives native notifications instead of application-specific ones.

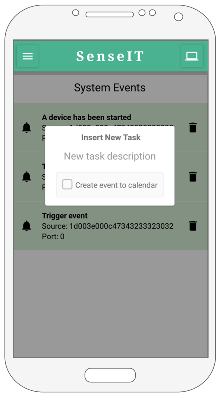


Name: Particle Photon

ID: 1d003e000c47343233323032

Defined rule:
Sensor Name: Touch with led
Digital 0 triggers Digital 7

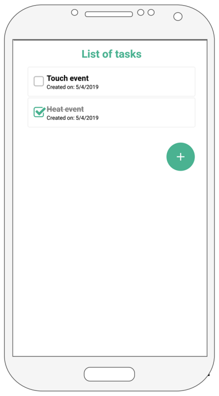
Fig. 5. Configuration Page



A device has been started

Trigger event
Source: 1d003e000c47343233323032
Port: 0

Fig. 6. Event Page



Touch event
Created on: 5/4/2019

~~Heat event~~
Created on: 5/4/2019

Fig. 7. Tasks Page

The server management tab lists the name and the IP address of any previously connected servers, and provides the possibility of discovering new ones. When the application is connected to a central server, the tab lists its available microcontrollers in the network, together with their IP addresses. By selecting a device, the user can reach the Device configuration page (see Figure 5).

Sensor rules are created on the settings page. The app distinguishes between two type of rules: trigger events and reminder events. To create a trigger event, an input and output GPIO pin must be selected. Reminder rules only have and input rule property, sending a notification to the client when the event is triggered, so the user is able to create a task from it. The application allows saving assignments from every notification entry, not just reminder rules.

V. Performance Measurement

A critical factor of any IoT system is the latency between the sensor input and the system reaction, even under heavier loads. This section outlines and demonstrates a performance measurement evaluation of the project.

For testing, the Raspberry Pi slave server generates changing numbers of events every second, simulation different levels of burden. The Pi attaches the generation time as the description of the created entry. As mentioned above, the master Node.js server polls the event list from the online microcontrollers, and saves event models in the database. With the help of *mongoExport*, the created entries are dumped and analyzed as JSON files.

The histograms described below are linked to 3 different experiments, all showing the distribution of elapsed time between event generation and persistence into the database.

In the first experiment (see Figure 8), the Raspberry Pi generates 300 events per second for 13 seconds. The average latency is measured as emph159 ms, showing an inferior performance than push notifications would be capable of. Analyzing an overlayed deduced normal curve, with reaction times rarely reach above 200 ms.

Figure 9 presents the system in a slightly overburdened scenario, with 600 events a second for 3 seconds. The average

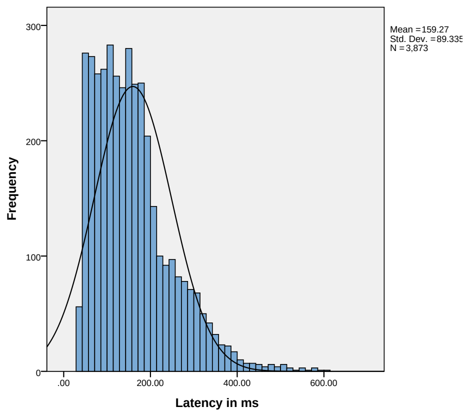


Fig. 8. Latency when processing 300 events a second

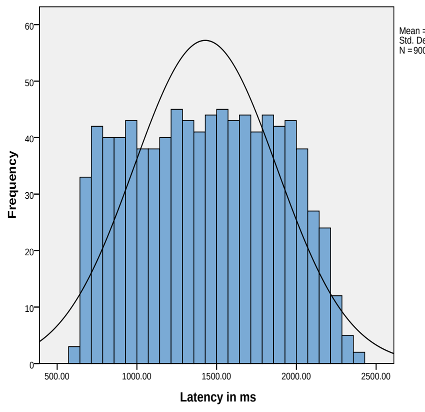


Fig. 9. Latency when processing 600 events a second

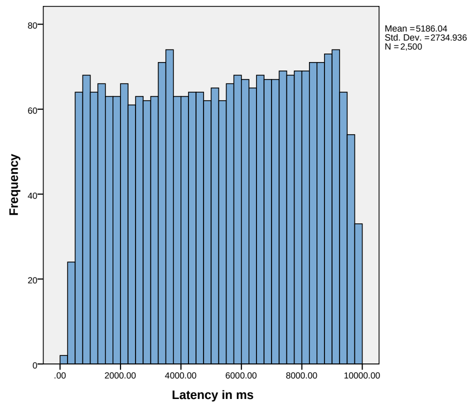


Fig. 10. Latency when processing 1000 events a second

latency increases to *1500 ms*, with values not following a standard normal distribution anymore. This amount of events strains the system enough to disallow real-time reading, saving and polling.

Finally, Figure 10 demonstrates the system in a highly overloaded situation, with 1000 events per second for 3 seconds. The average latency rises to *5187 ms*, showing a tendency to further increase with time, as an example of bottlenecking. A solution for decreasing the latency in the case of this amount of data could be to not store the events in the database.

VI. Conclusions and Future Work

The SenseIT project offers customers an all-in-one digital sensor monitoring system even in remote locations with no Internet access. The main purpose of the project is to build an IoT platform with changeable sensor sets for digital businesses. With the aid of the Particle Photon and Raspberry Pi microcontrollers, small companies can build a personalized IoT platform for monitoring their production workflow. Although multiple components and platforms are involved, the project manages to minimize the configuration process by using a self-advertising network protocol.

The system has been successfully presented as a general monitoring system, with usage examples drawn from and tailoring made specifically for the apiary industry branch. A mobile user interface using modern cross-platform technologies has been created for entrepreneur ease of use, not targeted solely for qualified software engineers.

The project has been tested and presented only with the help of the Particle Photon and the Raspberry PI microcontrollers. However many other devices exist on the market, which could be included into the project. A new MCU can be added by implementing or adapting one of the existing REST API implementations to communicate with the central server.

As another potential improvement, we propose extending the project with analog sensors for higher resolution information gathering. To derive usable data from analog sensors, conversion function(s) would need to be provided by users. E.g. in case of a temperature sensor the software would have to convert the measured value from the range of 0 to 5 volts to Celsius or Fahrenheit.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communication Surveys & Tutorials*, 2015.

[2] K. Schwab, *The Fourth Industrial Revolution*. Crown Publishing Group, 2017.

[3] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[4] B. Eisenman, *Learning React Native: Building Native Mobile Apps with JavaScript*. O'Reilly Media, 2015.

[5] M. Cantelon, M. Harter, T. Holowaychuk, and N. Rajlich, *Node.js in Action*. Manning Publications, 2017.

[6] Particle Photon Official Documentation. [Online]. Available: https://docs.particle.io/photon/

[7] D. Guinard and V. Trifa, *Building the Web of Things: With Examples in Node.js and Raspberry Pi*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2016.

[8] M. Hung, "Leading the IoT," Gartner, Tech. Rep., 2017.

[9] P. Rosenkranz, P. Aumeier, and B. Ziegelmann, "Biology and Control of Varroa Destructor," *Journal of Invertebrate Pathology*, 2009.

[10] K. Hamdan, "Crystallization of Honey," *Bee World*, vol. 87, no. 4, pp. 71–74, 2010.

[11] R. T. Fielding and R. N. Taylor, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[12] S. Cheshire and M. Krochmal, "Multicast DNS," Internet Requests for Comments, RFC Editor, RFC 6762, February 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6762.txt

[13] MongoDB Official Documentation. [Online]. Available: https://www.mongodb.com

[14] M. Satheesh, B. J. D'mello, and J. Krol, *Web Development with MongoDB and NodeJS*. Packt Publishing Ltd, 2015.