Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 109C (2017) 1092–1097

Procedia Computer Science

www.elsevier.com/locate/procedia

Architecture for embedded software in microcontrollers for Internet of Things (IoT) in fog water collection

José Fernando Mendoza[a], Hugo Ordóñez[a], Armando Ordóñez[b], Jose Luis Jurado[a]

[a] *Universidad de San Buenaventura, Cali , Colombia*
[b] *University Foundation of Popayan, 5St 8-58, Popayan, Colombia*

Abstract

This paper presents a software architecture for micro-controllers based solutions that run in capture data cards for Internet of Things (IoT). The present approach describes the components of the software architecture and its interaction. Equally, the architecture allows the development of modular and configurable applications as is focused on the overall design and system specification. The evaluation was performed in a Fog Water Collection system.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: Microcontrollers, software architecture, Internet of things, Sensors

1. Introduction

The shortage of clean drinking water in some places is generating several environment, health, and social problems. This situation occurs because many water sources are contaminated or exhausted [1]. Due to this shortage, many remote and rural populations have had to rely on water carts or wells that don't provide the price, quality, and quantity needed by people. As an alternative to this problem, the fog water collection (FWC) makes it possible to obtain water in dry or remote zones [2] by condensing the fog into droplets of water [3]

On the other hand, the increasing growth of the Internet has led to the interconnection of more and more devices, thus giving birth to the Internet of Things (IoT) [4]. IoT refers to an ecosystem in which devices such as sensors, smartphones, and household appliances are connected to a global network and can be monitored and managed from anywhere [5]. These IoT devices vary widely in use and features but are mainly supported by hardware micro-controllers. These microcontrollers use embedded software to performs all arithmetic-logic operations to gather, process and send data [6]. In this context, software engineering for microcontrollers can contribute to creating more robust, scalable and maintainable IoT applications [7].

Based on the above, this article proposes a software architecture for micro-controllers based solutions

Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2017.05.395

(Embedded Systems). This architecture describes the interaction between the software and the hardware components that coexist in the electronic data capture cards. The proposed architecture is used in a FWC system.

The rest of this paper is organized as follows: Section 2 exposes the motivation of the work and Section 3 details the proposed software architecture. Later in section 4 the evaluation is described. Finally, the conclusions and the future work are presented.

2. Motivation

Fog contains tiny droplets of suspended water that are condensed when they come into contact with solid bodies that intercept them. These FWC systems have several advantages, firstly, they don't require electric energy to extract or transport the water, besides the risk of contamination is low compared with other techniques. Therefore, FWC systems constitute a viable alternative in mountain regions where fog is frequent. However, for the proper operation of FWCs, some variables need to be monitored on the site, and it is here where IoT appears as an alternative[8].

Regarding reference architectures or guidelines, few reference architectural models for embedded software in the IoT can be found in state of the art [9],[10]. These existing models don't detail the architecture or the software components necessary for developing these solutions. Thus, there are not standard guidelines or reference models for building solutions for micro-controllers, and each particular solution must define its own architecture depending on the domain.

3. Proposed software architecture

The architecture proposed here is divided into layers to model the whole software solution that runs on the microcontroller. This architecture includes the structure, operation, and interaction of the components of an embedded system in a data capture card, each of these components are described below [9]

3.1. Quality attributes considered for the design of the architecture

Quality attributes consider functional and non-functional requirements such as performance, interoperability, usability, flexibility, maintainability, security and scalability [11],[12]. These quality attributes depend on the functional requirements and technical constraints of the domain (for the case study considered here, the domain is real-time capture and monitoring of climatic variables).

In addition to the quality attributes, other elements were considered during the design of the architecture such as the software maintainability [13]. This attribute enhances the team's communication and contributes to the rapid evolution of the product, the reduction in maintenance cost, and a better documentation [14].

3.2. Functional requirements

The architecture integrates the functional requirements of the IoT domain, as shown below:

- The System should capture data from sensors connected to analog or digital ports.
- The system must normalize the captured data by applying standardization algorithms depending on the sensor.
- The system must allow user to change the status (ON / OFF) of the actuators connected to ports.
- The System should be able to connect to the Internet using GSM / GPRS, Wifi or Ethernet.
- The system should be capable of sending the gathered data to a remote system using a communication protocol (HTTP, REST, CoAP, MQTT)
- The system must provide a configuration interface (mobile or web).

3.3. Architectural style

The architecture is composed of different layers and architectural components and is based on the pattern model-view-controller-communication [12]. These layers interact with the hardware components and provide a software-level abstraction that represents both the business logic and the hardware elements as can be seen in Fig 1.

3.4. Patterns included in the architecture

Fig 1 shows the layers and components of the architecture. The *Interface* layer enables the communication between the embedded system and the outside world. This layer implements a *Listener* (abstract class) that identifies the incoming requests, decodes the information and assign the request to the appropriate component (*Commands*). This layer implements the components depending on the source of the request that can be a serial port, or another type of port (The communication protocols are implemented in the *communication protocols* layer).

The *Commands* sub-layer implements all possible actions that can be triggered by incoming requests or events, these actions interact with the routines layer of the *domain controllers* or *Use case administrators*. These actions include the sensing of temperature, humidity, noise, radiation, among others. *Domains Controllers* or use-*case administrators* layers manage business logic, sensors, actuators, web client routines for data query on a remote server, and routines for setting the device in server mode (in this latter case the solution must include an interface or direct interaction between the user and device acting as a server).

The *model layer* includes all the logic that directly interacts with the low-level routines of the micro controller firmware. These low-level routines are implemented in the *system abstraction layer* in a communication component (COM). The COM component supports the communication between the microcontroller and the input-output ports.

The *System Abstraction Layer* is divided into sub-layers that implement low-level or service routines that support the business logic. To achieve this, the *Kernel* component use routines that enable the normal execution of the application in the micro controller, here the initial configuration routine (setup) must be implemented as well as an infinite cycle routine (loop). This latter loop can only be stopped by a scheduled event or power supply disconnection.

The *Communication Protocols* sub-layer, as its name indicates, this layer groups the routines that implement the protocols to communicate with the outside world. In this layer, it is possible to find protocols wrapped in HTTP such as REST, CoAP, or MQTT.

The *Communication sub-layer* implements the communication between the card and the exterior (internet). These routines of service and its configuration depend on each card and the implemented service. The *COM sub-layer* for low-level communication implements the exchange of data to the analog or digital communication ports of the micro-controller. This implementation enables the systems to interact with sensors, actuators or another external device (physical world) connected to the card. Finally, the *system layer* implements a sub-layer in charge of the *Local Store Service of local data* in case the application requires it.

4. Evaluation and results

The evaluation of the proposed architecture is divided into two parts, a) Selection of a methodology for the evaluation of software architectures, b) definition of the variables to be evaluated.

4.1. Evaluation methodology

The evaluation is based on the Architecture Tradeoff Analysis Method (ATAM), This methodology allows to evaluate Software Architecture according to the quality attributes specified for the system to be developed [15].

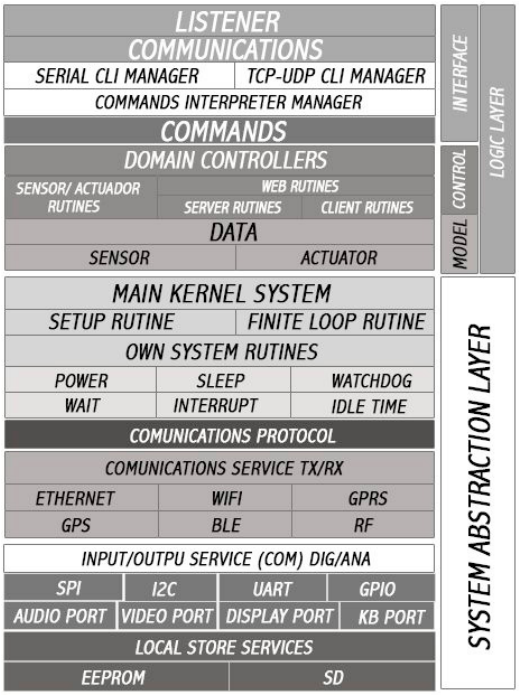

Fig. 1. Layers and components of the proposed architecture

ATAM provides a characterization for various attributes based on the existing knowledge of quality attributes such as performance, modifiability, availability and security.

Regarding performance, the architecture supports parallelism, decomposing work into different processes that can be executed in parallel and cooperate with each other. This functionality optimizes the communication between processes, the network use and the frequency of access to data. For example, the data can be retrieved from the mobile or Web application. Besides, it is also possible to capture data from several variables at the same time, such as temperature and humidity. The modifiability by its part allows decomposing the application in layers so that its development has greater modularity and scalability. Thus the portability of the developed solution increases. On the other hand, the security contributes elements to control, monitor and audit the actions that can be executed by the various components of the data capture card. Also, it provides the possibility of detecting and recovering faults during the data capture or transmission. Finally, the availability measures the capacity of use and execution of the software during intervals of time (one month). For example, in the case study, the capture of agroclimatic data was evaluated. During this period, the software evidenced stability and security. These results allowed us to claim that the proposed architecture complies with the quality attribute.

4.2. Definition of the variables to measure

The prototype was built using the proposed architecture to monitor temperature, relative humidity, and quantity of water collected. The prototype is part of the "*Implementation of an alternative technological prototype for the capture Of water using the fog*" project of the Latin American Center of Small Species CLEM. The prototype was built using C ++ and Atmel Studio programming environment. The software implements the layers described in Fig 1. The sub-layers of the ***System Abstract Layer*** interact with the hardware of the card (I/O ports, memory, eternal communication ***Kernel***) and the implementation of the HTTP protocol for the information exchange. Regarding ***Logic Layer***, the MVC pattern was used.

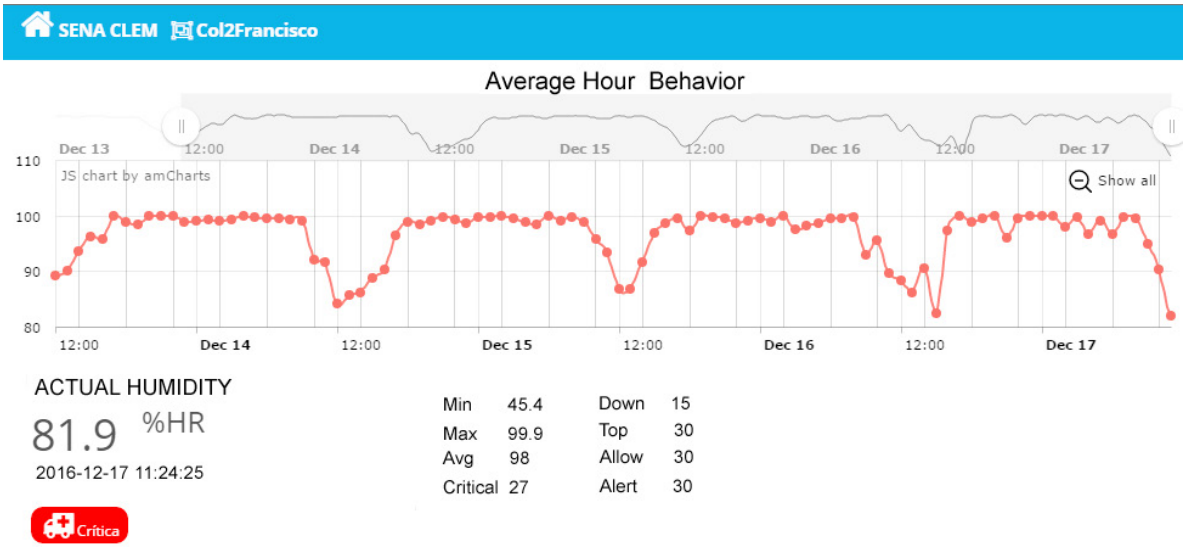

Fig. 2. Temperature and humidity – average hour to hour

The prototype used an IoT card MediaTeck's LinkIt One (Wifi, GSM), with an ARM7 EJ-S micro-controller, 32-bit RISC architecture, 260Mhz, 4M RAM and Flash 16M. All the C++ libraries of the provider are publicly available, and a rapid implementation is possible. Besides, these cards are easily accessible. The gathered data are managed using a Web application (OpenIO Enterprise) and a mobile App (OpenIO Mobile for Android 4.3 or higher shown in Fig. 2) that uses Web services to interact with the Database. With OpenIOMobile, the current state of the sensors can be known in real time, and some alerts are triggered when predefined thresholds (allowed/alerted / critical) are exceeded. The prototype measured temperature, relative humidity, and quantity of water collected an FWC system. Some behavior curves were generated to determine the best conditions for collecting water.

The prototype integrates a DHT22 sensor. This device consists of a capacitive sensor and a thermistor to measure temperatures (-40 to 80 °C, accuracy of ± 0.5 °C) and humidity (0% to 100%, accuracy of 2%) [26]. Equally, an Ultrasonic sensor SRF05 was used to measure the water level (3cm to 400cm). Knowing the height of the water inside a 200 liters tank makes is possible to calculate the volume of the water collected by each FWC.

Fig 3 shows the humidity, temperature, and volume of collected water (liters), between 5 pm and 8 am. The data were collected in the village of Montañitas (Yumbo, Valle del Cauca, Colombia). Motañitas is located 17 Km away from the municipal seat of Yumbo, in the Central mountain change, at the height of 1500 meters above the sea level. The average temperature of this location oscillates between 19 ° C and 23 ° C, the climate is warm-humid tropical, and there are winds from the Colombian Pacific with condensed water in the form of non-stationary circulating fog. The average relative humidity is between 80% and 100%. As can be seen in Fig 3, the average volume of collected water in each FCW is 87 liters per day. Between 1 and 4 a.m., the temperature reaches the lowest level (13.5 ° C) and the humidity reaches the highest level (99%). The water collected is potable as it contains a higher level of purity than rainwater, because of its low mineral content. Also, this water can be used in other agricultural activities.

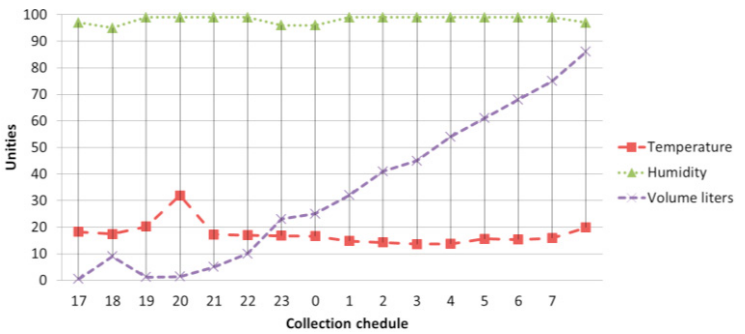

Fig. 3. Temperature, Humidity and collected water vs. Hours of the day

5. Conclusions and future work

This paper presents a software architecture for the development of embedded systems running on microcontrollers. The architecture doesn't depend on the specific domain and focuses on the functional requirements of the software, emphasizing on quality attributes. The layered design of the architecture provides the possibility of identifying the components that the software must contain and how each of them must interact with the elements of the data capture cards. The evaluation process allowed to identify that the use of the architecture makes it possible to comply with quality attributes such as maintainability, security, scalability, among others. During the evaluation, the quality of availability attribute is largely fulfilled because the data were captured consistently over a period of 2 months. On the other hand, the data capture enabled the CLEM - SENA researchers to perform their research regarding the manual of construction of the cloud collectors. Future work is oriented towards the evaluation of the proposed architecture in other domains, such as water quality measurement, river flow control, air quality management, etc.

References

1. Prada, S., Menezes, M., Sequeira, D., Figueira, C. & Oliveira, M. Agricultural and Forest Meteorology Fog precipitation and rainfall interception in the natural forests of Madeira Island (Portugal). **149,** 1179–1187 (2009).
2. Regalado, C. M. & Ritter, A. Agricultural and Forest Meteorology The performance of three fog gauges under field conditions and its relationship with meteorological variables in an exposed site in Tenerife (Canary Islands). *Agric. For. Meteorol.* **233,** 80–91 (2017).
3. Harb, O. M., Salem, M. S., El-hay, G. H. A. & Makled, K. M. Fog water harvesting providing stability for small Bedwe communities lives in North cost of Egypt. *Ann. Agric. Sci.* **61,** 105–110 (2016).
4. Ashton, K. That 'Internet of Things' Thing. *RFiD J.* 4986 (2009).
5. Dorsemaine, B., Gaulier, J. P., Wary, J. P., Kheir, N. & Urien, P. Internet of Things: A Definition and Taxonomy. *Proc. - NGMAST 2015 9th Int. Conf. Next Gener. Mob. Appl. Serv. Technol.* 72–77 (2016). doi:10.1109/NGMAST.2015.71
6. Lipovski, G. J. Introduction to Microcontrollers. *Introd. to Microcontrollers* 379–414 (2004). doi:10.1016/B978-012451838-4/50016-9
7. Jararweh, Y. *et al.* SDIoT: a software defined based internet of things framework. *J. Ambient Intell. Humaniz. Comput.* **6,** 453–461 (2015).
8. Quwaider, M., Al-Alyyoub, M. & Jararweh, Y. Cloud Support Data Management Infrastructure for Upcoming Smart Cities. *Procedia Comput. Sci.* **83,** 1232–1237 (2016).
9. Bauer, M. *et al.* Introduction to the Architectural Reference Model for the Internet of Things. *Internet-of-Things Archit. – IoT-A Deliv. D1.3 – Updat. Ref. Model IoT v1.5* (2012).
10. Misra, P., Simmhan, Y. & Warrior, J. Towards a Practical Architecture for the Next Generation Internet of Things. *Arxiv* 1–6 (2015). doi:10.1109/TIT.2016.2527683
11. Clements, P. & Kazman…, R. Evaluating software architectures: methods and case studies .
12. Bass, L., Clements, P. & Kazman, R. *Software Architecture in Practice*. *Vasa* **2nd,** (2003).
13. Graaf, B. Maintainability through architecture development. *Softw. Archit.* 206–211 (2004). doi:10.1007/978-3-540-24769-2_16
14. Clements, P. & Garlan, D. Software Architectures. 1–6 (2002).
15. Patidar, A. & Suman, U. A survey on software architecture evaluation methods. **49,** 967–972 (2015).