RASPBERRY PI AND WINDOWS 10 POWERED INTELLIGENT MODULAR GATEWAY FOR DECENTRALIZED IOT ENVIRONMENTS

Matej KVETKO, Jozef MOCNEJ, Ladislav POMŠÁR, Iveta ZOLOTOVÁ
Department of Cybernetics and Artificial Intelligence, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Slovak Republic,
E-mails: kvetkom@gmail.com, jozef.mocnej@outlook.com, ladislav.pomsar@tuke.sk, iveta.zolotova@tuke.sk

ABSTRACT

With an ever-increasing number of connected devices, parts of the decision process necessary for IoT environments are shifted from the cloud back to the local network. The onset of so-called Edge computing increases the demand for intelligent modular gateways. These gateways should be able to support a large number of different sensors, actuators, protocols, and applications. In this work, such a modular intelligent gateway for decentralized IoT environments is designed. The implementation of the gateway is based on Windows 10 IoT Core and Raspberry Pi 3. This gateway allows to plug-in sensors, protocols, and data processing applications in real-time, without the need for gateway restart.

Keywords: *Internet of Things, Intelligent modular gateway, Windows 10 IoT Core, Edge computing, Raspberry Pi*

1. INTRODUCTION

Over the last few years, the Internet of Things (IoT) has become part of our lives. For years, IoT has been closely tied to cloud computing. First, the sensors did measure values the user was interested in, send them over different protocols to the gateway that propagated the information into the cloud. Therefore, the gateway is acting as a bridge between the sensing domain (sensors and actuators) and cloud [1]. However, with the ever-increasing number of sensors, the networks have become increasingly stressed. Concurrently, due to the increasing number of sensors, the IoT infrastructure planning has become increasingly problematic as the infrastructure needs to be flexible. The networks are, therefore, becoming the weak points of the entire IoT system [2]. Also, the demands of the users have increased, resulting in new challenges in data processing. Aside from these problems, problems with security, privacy, access, latency, reliability, remoteness etc. arose.

Edge computing is aiming to address some of the mentioned problems. It is providing the computations and decision making closer to the source - on the edge of the network, while some of the logic still resides in the cloud. We call this kind of architecture a decentralized architecture. Edge computing may take on some of the functionality that was firstly addressed by the cloud or just preprocess/filter the data for the cloud. This way, edge computing can reduce the amount of transferred data. While decentralized architectures scale easier with an increasing number of sensors, they increase the complexity of the decision process [2].

To address this problem, in this article, we propose an intelligent modular gateway for a decentralized IoT environment to address the management complexity. The rest of the paper is structured as follows - section 2 outlines the background and related work. The architecture design proposal is analyzed in section 3 and section 4 describes the implementation process of developing a modular gateway on Raspberry Pi 3 with Windows 10 IoT Core.

2. RELATED WORK

There are several academic and industrial works concerned with IoT gateways. Zhu et al. [3] have designed a gateway architecture that was the center of wireless sensory network (WSN) in the smart-home application. The gateway itself was merely a connection between sensors and actuators that did not have an IP address and applications that required data from the sensors. The gateway itself was not scalable and could only serve certain applications.

In work presented by Datta et al. [4], a wireless gateway, that allows user equiped with mobile phone to display sensory data and control actuators, was divided into a northern and southern interface. The northern interface actedwas utilized to communicate and discover mobile clients, while the southern interface acted as an interface to manage, interact and store configuration of M2M devices in a database. The gateway itself is an M2M communications agent between smart devices and sensors. Such communication takes place only via the REST-API. This means that devices that do not support the HTTP protocol must be linked through additional gateway to provide the protocol. Metadata must be sent as soon as the device is first connected.

In the work of Guoqiang et al. [5], they presented a smart gateway that enabled a modular solution for connection interfaces. The interfaces themselves were handled with user cards. These cards were replaceable in slots so that the user could configure the gateway to their requirements. This results in only a limited number of connectable interfaces.

The work of Lin Wu et al. [6] is quite similar to ours. They have created a service-oriented gateway with plug and play configuration. It focused on a fast system that allows them to plug-in sensors easily. The gateway packs data, sensor capabilities, and sensor information into a homogeneous form. It provides external applications with an interface that is simple and understandable. However, the whole gateway works on the PC platform, so it is not ideal for the usage in the IoT environment.

Morabito et al. [7] designed so-called LEGIoT - a Lightweight Edge Gateway for the IoT. Their solution utilizes virtualization technology via containers and microservices to create an edge gateway. The gateway itself consists of 2 modules - Northbound, which is responsible for the communication with the Internet and Southbound that communicates with sensors. All parts of their architecture are virtualized via Docker containers. LEGIoT aims to provide interoperability, energy efficiency, flexibility combined with the benefits of virtualization technologies. They also provide concrete implementation on several single-board computers.

Kesevan and Kalambettu [8] introduced universal plug and play gateway for smart health. The core of their architecture consists of device service that is responsible for the management and gathering of the data from BLE/Wifi devices. The Data Manager is responsible for receiving and storing the data. The Network Manager, on the other hand, handles the data sending to the cloud and offline data sync. The work is only the proposal without a concrete implementation.

To the best of our knowledge, this work is one of the first works proposing service-oriented, plug and play Windows 10 IoT Core powered smart modular gateway with concrete implementation on hardware.

3. ARCHITECTURE DESIGN

The most significant benefit of a modular gateway is that it allows to aggregate data according to a user-selected scheme. Input data should serve as a basis for calculating new data that will then be sent to the cloud, visualized by the user, or sent directly from another device on the network. The user should be able to define what data they want to see. The system should be able to aggregate schemes created by the user. In this section, we will provide an insight into the process of architecture design.

3.1. Requirements

When creating the architecture, we had several requirements in mind:

1. The architecture is designed so that it is possible to create a modular system with dynamically interchangeable components for communication with sensors, actuators, and the cloud.
2. It should be possible to dynamically connect and disconnect applications from the pipeline while keeping the pipeline fully functional. The same requirement should be met for different communication protocols.
3. Architecture should be usable on different types of devices
4. Gateway should be able to filter and direct the incoming data

To meet the first two design requirements, we decided to design applications for the communication of sensors and actuators as services. These can be connected or disconnected if necessary, without affecting different parts of the system. Therefore, it is also easy to add a new protocol or interface to the gateway without the whole gateway being turned off just by creating new microservice utilizing this protocol. The third was met via the utilization of the Windows 10 IoT platform. To filter the data, it was also necessary to evaluate the fitness of input data.

3.2. Evaluation of data fitness

As the primary function of the modular gate is to aggregate data based on a user-defined schema, input data are further processed. To understand whether data are worth processing, we had to define data fitness criteria. There are two phases of assessing data fitness. The first phase takes place in sensors - the damaged/wrongly measured data are filtered out and aren't send to process in the gateway. The second phase takes place in the gateway itself. In this paper, we will work with two quantities that describe the data fitness - Quality of Information (QoI) and Value of Information (VoI) as defined by [9]. We will also assume that network is stable, with zero latency and constant access time for any of the sensors.

3.2.1. Quality of information

To make it easier to decide which data is suitable for further use, it is advisable to assign certain properties in the form of metadata to individual values. These metadata are different for every sensor. Most of the metadata assigned to the sensor are immutable - they are defined by the manufacturer in the datasheet and aren't expected to change. Subsequently, we can select the data with the best QoI, exclude data that does not meet certain qualities, or manage the collection of data by sensors according to the metrics we choose. [9] In general, metadata can be stored and sent directly from the sensor or assigned to the virtual sensor later in the gateway. We have resorted to the second option due to three reasons. Firstly, some of the sensors have barely enough memory for their operation and storage of complex metadata would constraint their usage. Secondly, some of the protocols have constrained message size - for example, BLE allows for 20 bytes of attribute data. Thirdly, sending metadata periodically in every message would drastically increase the volume of data processed by the gateway. With simple values, like temperature readings, metadata may also be several times larger than the value itself. In the ideal case, the gateway would use an external database that stores all metadata for all potential sensors. However, this is not yet possible, and therefore data were statically stored in the gateway. Metadata used in our work were:

1. The cost of measurement - Evaluates the financial cost of measurement
2. Accuracy - Supplied by the manufacturer. The value itself consists of accuracy, range, where the accuracy applies, and the flag indicating whether accuracy is absolute or relative value
3. Resolution - Supplied by the manufacturer. Describes the smallest change of quantity sensor can measure

4. Range - Supplied by the manufacturer. Describes the maximum and minimum values of the quantity sensor can measure

3.2.2. Value of information

For this paper, we decided to follow the definition of VoI defined by Beskidian et al. [9]: "Value of information (VoI) is an assessment of the utility of an information product when used in a specific usage context." Value of information is a quality value that differs for each information and for each application that processes the data. It is, therefore, necessary to evaluate it separately case by case.

Most of the time, several parameters are selected to asses the VoI. In this work, we utilized the range for all the sensors and combined range and accuracy for some of the sensors. In general, if the value is outside the range stated by the manufacturer, the whole measurement is deemed as wrong. If there were several sensors with the same measured quantity and range couldn't select the best one, we did use accuracy. Some of the sensors have relative accuracy, while others have absolute accuracy. Depending on the current values, some may measure more accurate values than others. The more accurate sensor has higher VoI and its reading is visualized to the user. There is no need for the user to know what sensor did measure the value as we are always displaying only the fittest one.

4. IMPLEMENTATION

The architecture is designed so that it is possible to create a modular system with dynamically interchangeable components for communication with sensors, actuators, and the cloud. It should also be possible to dynamically and at the full functionality, add and remove applications that are connected to the pipeline, and which will take care of the data aggregation.

Sensor and actuator applications for communication are designed as stand-alone applications. These can be connected or disconnected if necessary, without affecting a separate system. Therefore, it is also easy to add a new protocol or interface to the gateway without the whole gateway being turned off. Everything can be done during the full operation of the gate. Applications connected to pipelines work in a similar way. They receive pipeline information and re-insert new information into the system as new input data.

4.1. Software

As one of the requirements was to support different types of devices, we had to evaluate software possibilities. As we had good experiences with Microsoft technologies, we resorted to Windows 10 IoT Core as an operating system. Windows 10 IoT Core [10] is a version of Windows 10 optimized for smaller x86/x64 and ARM devices. These devices may or may not feature display. In general, there are several differences between Windows 10 and Windows 10 IoT Core:

1. Since version 1809, IoT Core does lack Cortana
2. Windows 10 IoT Core does allow to boot into predefined application
3. Some APIs and drivers aren't supported
4. Some registry entries and shell commands behaves differently.

4.2. Hardware

As Windows 10 IoT Core was selected for the operating system, we had to evaluate different types of single board computers usable with this operating system. The overview is visible in Table 1.

Aside from displayed single board computers, Raspberry PI 3B+ (RPi 3B+) was also available at the time of the architecture implementation. However, Windows 10 IoT Core support for RPi 3B+ was only experimental, did lack some of the drivers, and had other problems. Due to these problems, RPi 3B+ was excluded.

Due to availability, price, community support, and capabilities, we selected the Raspberry Pi 3B (RPi 3B) [18]. RPi 3B is a successor of popular RPi 2. It features quad-core 64 bit ARM CPU, 1 GB of RAM, and SD Card slot within 85 x 65 mm form factor. As of connectivity, it features ethernet port, 4x USB 2.0 port, Wi-Fi 802.11b/g/n, Bluetooth 4.0/LE, 40 GPIO pins, 3.5 mm jack, and HDMI port.

For testing purposes, we utilized a set of 5 sensors - DHT11, BMP180, DS1820, YL-96, and LM-396.

4.3. Architecture overview

In our work, we have been inspired by the work of Mocnej et al. [2], which highlights the universal architecture of modular gateway. While the highlighted architecture is operation system agnostic, in our work, we were oriented on Windows-specific implementation of highlighted principles. The architecture is visible on the image Figure 1. A brief overview of different parts:

1. **Data input and cleaner** - Application responsible for standardization of input data. In case some of the input values aren't available/valid, the input is labeled as invalid and thrown away. The only exception is the date and time of measurement, as some of the protocols don't support the provision of these values.
2. **Metadata Manager** - Responsible for associating of right metadata with incoming data. Every incoming data should have property stating who did generate the measurement and therefore metadata manager can match it with its database
3. **Data Querry** - As two former applications are running in several parallel processes, some of the operations in parallel can cause the system to fail (e.g., several concurrent threads loading the same file). To avoid this, a single instance of a first-in-first-out buffer application data query is created. Data Query feeds data to destination selector
4. **Destination Selector** - This application is responsible for forwarding information to the selected receiver. Every single data coming to the pipeline

Table 1 Overview of single board computers supported by Windows 10 IoT Core

Series	Model	Wi-Fi	Bluetooth	Ethernet	Video output
AAEON	Up Squared [11]	No	No	2x	HDMI/DP
Dragon Board	410c [12]	802.11b/g/n	4.1/BLE	No	HDMI
MinnowBoard	Turbot [13]	No	No	1x	HDMI
NXP	i.MX6 [14]	No	No	1x	Paralel
	i.MX7 [15]	No	No	1x	No
	i.MX8M/Mini [16]	No	No	1x	HDMI
Raspberry Pi	2 [17]	No	No	1x	HDMI
	3B [18]	802.11b/g/n	4.0/BLE	1x	HDMI

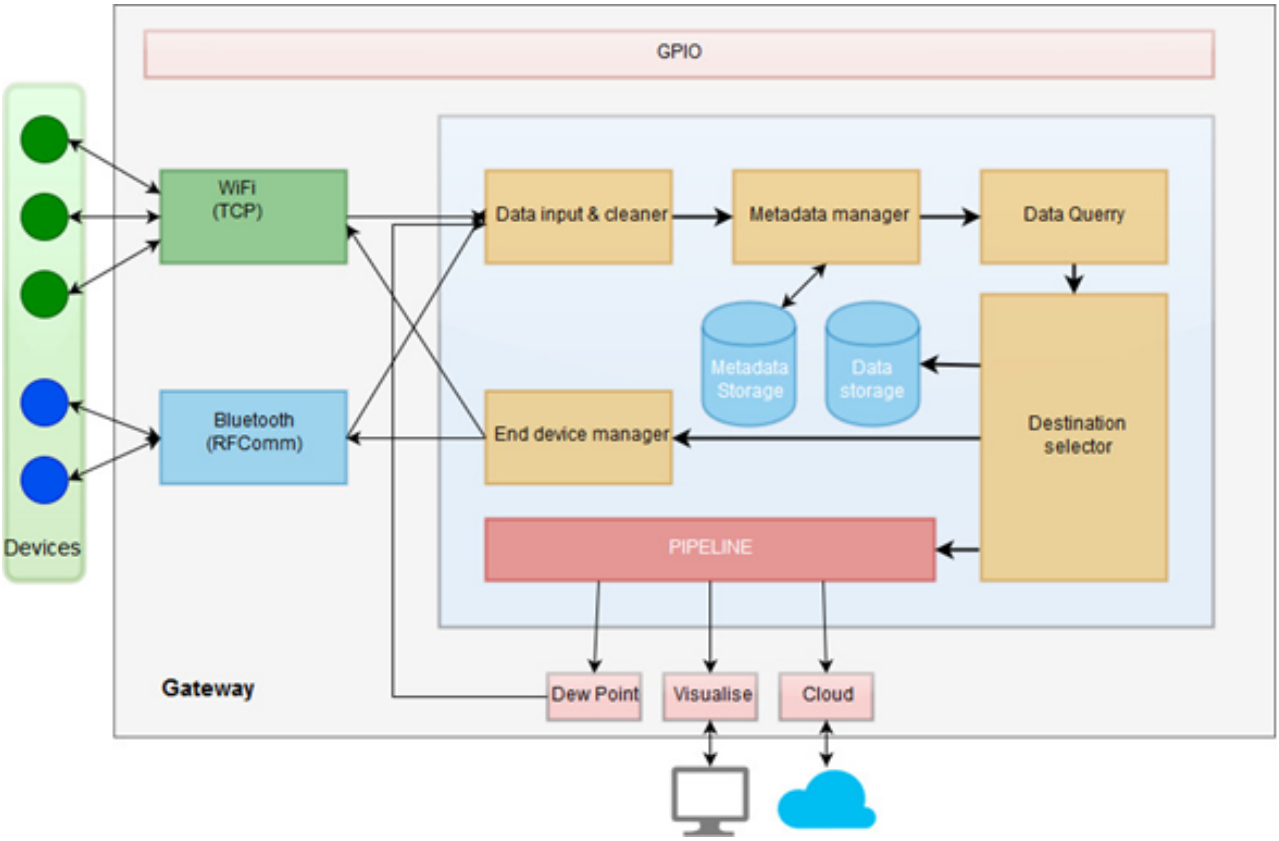

Fig. 1 Implemented architecture of the gateway

should have a defined receiver. In the current implementation, if this isn't true, every single application decides whether it can use data. QoI and VoI based filtering are also implemented here.

5. **Pipeline** - Consists of services. Responsible for bringing the data to connected applications.

6. **End Device Manager** - Responsible for sending data through the different protocols back to connected devices, if needed.

Data input and cleaner roughly equals the Connectivity and Connectivity abstraction layers of the reference architecture. Metadata manager, Data Query, and Destination Selector together create our Gateway services layer. Gateway services layer is also partially supported by pipeline, which also takes on the responsibilities of the Service management layer.

4.4. Pipeline and application services

To receive and distribute data from/to data processing applications in our architecture, we use the pipeline. The pipeline works on the basis of application services in Windows 10 IoT Core. The services were added to Windows 10 in version 1607. Since version 1703, it is also possible for a single application to utilize multiple services. In general,

service is a special type of application that run in their own Windows session and do not show any user interface.

There is a single problem with the pipeline-based approach. The background service and tasks are the same type of application. The same deferral is used to manage and terminate them. In practice, this means that the data processing application creates a service instance, discards it by deferral after it is used. Since the entire background application uses the same deferral, it will also discard the entire application. Although the application will restart, it will lose the information stored in the local memory. This is an extremely unwanted behavior. Therefore, it is necessary to divide each application into two separate units. One takes care of receiving information, and the other part called the main application takes care of sending the data. Even unsuccessful sending of information does not cause instability or closure of the application.

First, applications must log in to the pipeline at the start. This happens at the start when the application writes its unique name (i.e., the string that is in the static class of Package Windows.ApplicationModel.Package.Current.Id.FamilyName) to the shared file. Windows 10 IoT Core allows sharing files only in specific system files. For internal reasons, it is not possible to use the Documents folder or the system disc C://. All unique application names that want to be attached to the pipeline are stored in this folder. In this case, the pipeline / pipeline.txt file.

In case of a connection error with the application, the pipeline itself ensures that the application is deleted from the file and vice versa, if the application is registered to the file, the pipeline tries to connect to it automatically without the user's intervention. In the current settings, the file is being checked before each arrival of data into the pipeline. Obviously, this is not necessary, and in the future, it is advisable to check the connected applications every few seconds and to connect them regardless of whether data come. The connection itself lasts several tens of milliseconds and doesn't cause any performance overhead. However, if a slow SD card is used, the total time of processing a single message in the pipeline can increase to several hundred milliseconds. The amount of apps connected does not greatly affect the time it takes to send information - it is more or less constant.

4.5. Receiving and sending the information inside gateway

Every single data application is split into two applications - background service that can receive and process messages and main application.

The background service picks information in the form of key-value pairs and provides an answer to the sender. In case the data are not picked up, the *TimeOut* exception is returned. The received value is stored in the service application, and it needs to be sent to the main app. This is done using static *ApplicationData* storage. *LocalSettings* is used to share data between individual components of one application. The service then notifies the main application via an event that the new data are available in *ApplicationData* and should be picked up. Further processing in the main application depends on the logic of the application.

Sending is more straightforward than receiving because the main application itself can do it. For sending, it is necessary to know the unique name of the application for which the data are intended. Since there are often multiple applications, we had to create a system that automatically obtains and uses those applications' names. As mentioned before, we will use a text file to load the applications' names and then try to connect to them.

Subsequently, we need to save the service reference so that we do not have to connect to it every time gateway sends information, but just call it and send data. In case of successful sending of data, a confirmation will be returned. After the time-out has elapsed or an error has occurred, an error is returned. This means that the service responsible for receiving messages has a problem, so it is not appropriate to send it additional data. We, therefore, delete it from the database. If it ever regains full functionality, it will log into pipeline again.

5. CONCLUSION AND SUMMARY

In this paper, we designed a modular intelligent gateway. This gateway aims to provide a link between sensors and actuators on one side and both local and cloud applications on the other. The gateway aims to reduce the amount of data it sends to applications based on the quality and value of the information. The entire gateway was implemented on windows 10 IoT Core powered RPi 3 via the usage of application services. Services were utilized to ensure the modularity of applications attached to the pipeline and helped to apply the proposed architecture. Thanks to the application service, individual components can be switched on and off without any altering the pipeline runtime. There are several possible directions to continue our work. Firstly, it is necessary to validate our work in a real-life environment with many different sensors connected and design stress testing methodology to asses the abilities of the gateway. Secondly, our current setup does not feature any edge AI accelerator or other computing platforms.

ACKNOWLEDGEMENT

This work was part of the Matej Kvetkos master thesis. This paper was supported by the grant KEGA 1/0663/17 - AICybS - Smart Industry/Architectures of Intelligent Information and Cybernetic Systems.

REFERENCES

[1] H. CHEN, X. JIA, and H. LI, "A brief introduction to IoT gateway," in *IET International Conference on Communication Technology and Application (ICCTA 2011)*. IET, 2011, pp. 610–613.

[2] J. MOCNEJ, W. K. SEAH, A. PEKAR, and I. ZOLOTOVA, "Decentralised iot architecture for efficient resources utilisation," *IFAC-PapersOnLine*, vol. 51,

no. 6, pp. 168–173, 2018.

[3] Q. ZHU, R. WANG, Q. CHEN, Y. LIU, and W. QIN, "IOT Gateway: BridgingWireless Sensor Networks into Internet of Things," in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, 2010, pp. 347–352.

[4] S. K. DATTA, C. BONNET, and N. NIKAEIN, "An IoT gateway centric architecture to provide novel M2M services," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 514–519.

[5] S. GUOQIANG, C. YANMING, Z. CHAO, and Z. YANXU, "Design and Implementation of a Smart IoT Gateway," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, 2013, pp. 720–723.

[6] L. WU, Y. XU, C. XU, and F. WANG, "Plug-configure-play service-oriented gateway-for fast and easy sensor network application development," in *International Conference on Sensor Networks*, vol. 2. Scitepress, 2013, pp. 53–58.

[7] R. MORABITO, R. PETROLO, V. LOSCRI, and N. MITTON, "LEGIoT: A lightweight edge gateway for the Internet of Things," *Future Generation Computer Systems*, vol. 81, pp. 1–15, 2018.

[8] S. KESAVAN and G. K. KALAMBETTU, "IOT enabled comprehensive, plug and play gateway framework for smart health," in *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC)*. IEEE, 2018, pp. 1–5.

[9] C. BISDIKIAN, L. M. KAPLAN, and M. B. SRIVASTAVA, "On the quality and value of information in sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 9, no. 4, pp. 1–26, 2013.

[10] TerryWarwick, "Overview of Windows 10 IoT Core - Windows IoT," May 2020, [Online; accessed 11. May 2020]. [Online]. Available: https://docs.microsoft.com/en-us/windows/iot-core/windows-iot-core

[11] "World's Fastest Maker Board_UP Squared," May 2020, [Online; accessed 12. May 2020]. [Online]. Available: https://www.aaeon.com/en/p/iot-gateway-maker-boards-up-squared

[12] "DragonBoard™ 410c," May 2020, [Online; accessed 12. May 2020]. [Online]. Available: https://www.96boards.org/product/dragonboard410c

[13] "MinnowBoard Turbot - MinnowBoard Wiki," Jan 2020, [Online; accessed 12. May 2020]. [Online]. Available: http://minnowboard.outof.biz/MinnowBoard_Turbot.html

[14] "i.MX 6 Series Applications Processors | Multicore Arm Cortex-A7/A9/M4 | NXP," May 2020, [Online; accessed 12. May 2020]. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-6-processors:IMX6X_SERIES

[15] "i.MX 7 Series Applications Processors | Arm® Cortex®-A7, Cortex-M4 | NXP," May 2020, [Online; accessed 12. May 2020]. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-7-processors:IMX7-SERIES

[16] "i.MX 8 Series Applications Processors | Arm® Cortex®-A72/A53/A35/M4 cores | NXP," May 2020, [Online; accessed 12. May 2020]. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-8-processors:IMX8-SERIES

[17] "Buy a Raspberry Pi 2 Model B – Raspberry Pi," May 2020, [Online; accessed 12. May 2020]. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b

[18] "Raspberry Pi 3 Model B+," May 2020, [Online; accessed 11. May 2020]. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus

Received June 2, 2020, accepted June 15, 2020

BIOGRAPHIES

Matej Kvetko graduated (MsC) in Intelligent systems from the Department of Cybernetics and Artificial Intelligence, Faculty of Electrical Engineering and Informatics, the Technical University in Koice in 2019. Since then to 4/2020, he was working as Software developer in Siemens Healthineers and now he is Developer in Tieto Evry. His main focus is on Cloud technologies, Internet of Things (IoT) and Web developement.

Jozef Mocnej was born on 28.08.1992. He graduated (MSc) from the Department of Cybernetics and Artificial Intelligence, Faculty of Electrical Engineering and Informatics, the Technical University in Košice. Since 2015, he has been pursuing a Ph.D. at the same university. His research is related to the Internet of Things (IoT), Wireless Sensor Networks (WSN), and IoT architectures.

Ladislav Pomšár graduated (MsC) in Intelligent systems from the Department of Cybernetics and Artificial Intelligence, Faculty of Electrical Engineering and Informatics, the Technical University in Košice in 2019. Since the same year, he has been pursuing a Ph.D. at TUKE in collaboration with Siemens Healthineers. His main ressearch areas are AI in health diagnostics and AI in Edge.

Iveta Zolotová graduated from the Department of Technical Cybernetics of the Faculty of Electrical Engineering, Technical University of Košice, Slovakia in 1983. She defended her CSc. in the field of hierarchical representation of digital image in 1987. Since 2010 she has been working as a Professor at the Department of Cybernetics and Ar-

tificial Intelligence, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia. At the department, she is also the head of the Intelligent Cybernetic Systems research group . Her scientific research is focused on Smart Industry/Industry 4.0, Internet of Things, intelligent systems, networked control and information systems, supervisory control, data acquisition, human machine interface and remote labs. She also investigates issues related to vision and robotics.