Advanced sensors network in a centralized IoT system using low-cost microcontrollers and automatic configuration

Andrei-Mihai VĂDAN, Liviu-Cristian MICLEA

Department of Automation, Faculty of Automation and Computer Science,
Technical University of Cluj-Napoca

andrei.vadan@hotmail.com, liviu.miclea@aut.utcluj.ro

Abstract: Smart homes have become more popular recently and a lot of day-to-day activities are becoming automated. This paper describes a smart IoT system that self-tests the configuration and initializes the sensors connected accordingly, in case of a system designed for smart homes. The system consists of multiple gateways, microcontrollers, and sensors. It uses popular hardware, like Raspberry Pi, Raspberry Pi Pico, Espressif, Banana Pi, Nvidia Jetson and communication between microcontrollers is done in an ethernet network, with devices connected via wires or wirelessly. The system includes different kinds of sensors, like temperature, humidity, ambient light, proximity, ultraviolet radiation, or gestures detection sensors, from different manufacturers. The paper describes a method to create a system for smart homes that is easy to configure by the end user, even while having a large spectrum of configurations. It is also easy to operate, cheap in implementation, and scalable, using well supported hardware by software communities.

Keywords: Smart network, IoT, Self-configure, Automation, Intelligent Control for Automation Systems.

Rețea avansată de senzori într-un sistem IoT centralizat utilizând microcontrolere cu costuri reduse și configurarea automată a acestora

Rezumat: Casele inteligente au devenit mai populare recent, iar multe activități zilnice sunt automatizate. Această lucrare descrie un sistem IoT inteligent care auto-testează configurația și inițializează senzorii conectați în consecință, în cazul unui sistem conceput pentru case inteligente. Sistemul este format din mai multe gateway-uri, microcontrolere și senzori. Folosește hardware popular, precum Raspberry Pi, Raspberry Pi Pico, Espressif, Banana Pi, Nvidia Jetson, iar comunicarea între microcontrolere se face într-o rețea Ethernet, cu dispozitive conectate prin cablu sau wirelles. Sistemul include diferite tipuri de senzori, cum ar fi senzori de temperatură, umiditate, lumină ambientală, proximitate, radiații ultraviolete sau senzori de detectare a gesturilor, provenind de la diferiți producători. Lucrarea descrie o metodă de a crea un sistem pentru case inteligente care este ușor de configurat de către utilizatorul final, chiar dacă are un spectru larg de configurații. De asemenea, este ușor de operat, ieftin în implementare și scalabil, folosind hardware bine susținut de către comunitățile de dezvoltare software.

Cuvinte cheie: rețea inteligentă, IoT, autoconfigurare, automatizare, sistem automat inteligent de control.

1. Introduction

The Internet of Things (IoT) is represented by systems that allow devices to be inter-connected and monitored over Internet. The strong concept evolution started a few years ago and now it is applied in various environments, like smart-homes, smart-cities, industrial environments or even medicine. An intelligent system requires some advanced functionalities, to be scalable, easy to control and monitor remotely by the end-user. The advanced capabilities of a smart-home system may include remotely setting temperature in certain rooms, adjusting shades, managing lighting, monitoring energy usage, overseeing irrigation systems, controlling various appliances, enhancing home security, ultimately leading to cost savings and increased comfort. The user can control the system using different devices, like smartphones, tablets, voice assistants, laptops, or even TVs.

There are a few IoT automation systems on the market, already including some well-known open-source ones like Home Assistant or OpenHAB. Some popular vendors have created an

https://doi.org/10.33436/v34i4y202402

ecosystem where the user can integrate some appliances or lights controls, like Samsung SmartThings or Lg ThinQ, for example. Other systems offer the end user an application based on subscription and hardware from different manufacturers, among the most popular being eWeLink or Tuya. Beside those options, there are several systems proposed by academic researchers like Ahmed et al. (2018) or Stolojescu-Crisan, Crisan & Butunoi, (2021). Davidovic and Labus, (2016) also proposed a system. At this point in time, there are multiple communication technologies, like Bluetooth, GSM, Lora, Wi-Fi or Zigbee and other protocols. The number of Wi-Fi devices that can be used with routers for consumers are limited to a few connected devices, usually around 30 to 50 devices (Martin, 2021). However, this limitation can be bypassed, by using different network configurations. In terms of communication protocols, there are several ones available, like Message Queuing Telemetry Transport (MQTT) or "If this, then that" (IFTTT) or custom ones using different web Application Programming Interfaces (APIs). Custom web APIs are not always the most secure ones, sometimes data is transmitted even in plain text. Most of the systems are vulnerable to attacks and the end user privacy can be easily violated, because most manufacturers are eager to stay one step ahead of the competition, by reducing costs in building safe devices over efficient ones, missing encryption, or keeping the research and development costs down. The proposed system offers a simple solution to auto-configure the gateways, by automatically detecting the sensors and the microcontrollers connected to it. The system uses a custom API, developed in Python which is running on a Raspberry Pi or similar microcontrollers. With small adaptations, the system can run successfully on any hardware with 512 MB of RAM, if there is a possibility to install Python 3. There are different sensors connected via Inter Integrated Circuit (I2C) (Campbell, 2021a) and Universal Asynchronous Receiver-Transmitter (UART) bus (Campbell, 2021b). Remote sensors are connected wirelessly, using smaller microcontrollers like ESP8266 connected to Raspberry Pi Pico or Raspberry Pi Pico W. The system does not have a limitation to these devices. It can work with any microcontroller that supports a Wi-Fi connection, a minimum encryption mechanism, and a programing language like Circuit Python or Micro Python, preferably.

While working on a method to detect the temperature sensors deviation in a real-world scenario, it was encountered an issue with the configuration of gateways and wireless sensors (Vădan & Miclea, 2022). Since the experiment required multiple gateways, and each gateway had different sensors attached to it, manual configuration was difficult. Adding new sensors required extra time to configure each model. Existing academic papers describe methods to create systems and generic methods of communications between gateways and mobile devices, but none describes an automatic way of connected sensors detection. Most systems are using a strict sensors list, especially commercial devices, while the proposed system uses a large, predefined list of sensors which can be updated via a small software update. In most systems, the user is required to configure the remote sensors and, in some cases, this process is overcomplicated or requires a certain level of know-how. The goal of the present paper is to describe a method to simplify the configuration of a smart-home system and to develop a system that is scalable, flexible, cheap to implement and maintain, all while maintaining a minimum level of security and privacy. It is desired to achieve these goals, by developing a system similar to what is currently described in most academic papers, while using sensors connected to I2C or UART interfaces, and a client-server architecture, using Python as main programing language.

As a feature improvement, it is intended to study if same methods can be applied for Improved Inter Integrated Circuit (I3C) or Serial Peripheral Interface (SPI) interfaces and include open-source MQTT and IFTTT integrations. Another improvement can be considered in the usage of HTTP 3.0 and easy pairing for remote sensors, by using both Bluetooth and Wi-Fi.

The second section of the paper presents the related work done by academic researchers in this domain. In the third section there are described the main hardware requirements in terms of sensors, microcontrollers and single board computers, a general topology diagram. In the fourth section there is described the actual work, while in the fifth section are presented the results. In the last section there are the conclusions.

2. Related work

In the field of home automation, there is a rich variety of protocols and technologies. Some of these technologies were developed by international institutions and represent the state-of-the-art, while others have been developed by private companies. Those technologies were developed to work in a wired or wireless environment. Each of them presents different advantages and disadvantages, depending on the deployed scenario. In academic papers, there are described these advantages and disadvantages, but also some of the most common use cases that can be found in an IoT system.

A comparison of automation systems proposed in different academic papers is presented by Stolojescu-Crisan, Crisan & Butunoi (2021). The authors describe the application, the types of microcontrollers and communication options for each system. In the same article, there is also a comparison between the most relevant open-source smart-home automation solutions. One of the fields describes the main features of smart-home solutions, while others are related to programing languages and technologies used. Similar information can also be found in Froiz-Míguez et al., (2018). A smart home system consists of multiple sensors and devices from which the data is retrieved and there should be various ways for the user to interact with these devices. In most cases, the user interacts with the smart-home system via a personal device or a gateway. One of the main purposes when designing the proposed system was to study if the Human-Machine Interface (HMI) of a smart-home system can be tested by applying the Locate, Execute, Expect design pattern described in (Vădan & Miclea, 2023), and if the design pattern is applicable for complex projects.

This design pattern has been previously used for testing the user interface of mobile applications, and the goal was to check if it is applicable also when testing API's and web applications. Because of the need of remote sensors, the overall system architecture is like a Fieldbus, as described in (Chen et al., 2023; Zhang et al., 2024) and in other similar papers available online. Overall, the system architecture is close to other systems proposed by academic entities. In Froiz-Míguez et al. (2018), the "Generic fog computing architecture" is displayed. Ożadowicz (2024) presented a simplified version of Fog computing architecture, while similar architectures were also described in multiple papers available online (Froiz-Míguez et al., 2018; Stolojescu-Crisan, Crisan & Butunoi, 2021). The proposed system's architecture is commonly used in academic papers, but also in systems developed by private companies. After reviewing different aspects related to the state-of-the-art there were identified important topics like the lack of a standard for smart-home automation, the wide range of technologies and communication protocols, the lack of security, the lack of transparency regarding user data saved to the cloud, inconsistent methods for system configurations, and the usage of only specific sensors. Some of the open-source systems include a wide range of sensors, but commercial systems force the consumer to acquire only specific products and, in some cases, also subscriptions to have access to different functions. Neither academic nor commercial solutions offer a method to automatically detect the sensors connected to a gateway.

Taking into consideration the previous statements, the system proposed in this paper was designed to employ the widely used hardware, and implement basic communications between sensors, gateways and consumer devices, by using a common technology and the existing Wi-Fi or Ethernet infrastructure. On the configuration side, the solution automatically configures itself, and the input required from consumer side is minimal. The solution is easy to install, maintain and, in the end, easy to test. The consumer can add new wireless sensors with ease and, in case a sensor breaks, the system continues to function without interruption. The sensors can be replaced or upgraded without the need of a new gateway. The security is increased by the fact that the system registers the devices when the pairing process happens, and the consumer is using a main gateway.

3. Materials and methodology

A smart-home system is designed to improve the quality of life. Before pandemic, most systems used the same sensors for each product, but for example in automotive industry, to deliver

a car, the manufacturers had to search for compatible hardware and quickly integrate it, to be able to deliver a complete car. The costs to make such changes during production are high. The integration of new hardware is a slow process, especially in the automotive industry. The proposed system was designed from the beginning to use multiple sensors for temperature and ambient lightning. On the hardware side, there is required a master-slave configuration, like a field bus approach. Starting from the top and going to the bottom there is required a router which facilitates the Wi-Fi connections for microcontrollers and the communication between user devices and the system. Another important component should be a master gateway, responsible for handling API requests. In the proposed system, this master gateway is represented by a Raspberry Pi microcontroller, or a similar one in terms of performance, like an Nvidia Jetson or Banana Pi. All other gateways are responsible for gathering data from sensors or performing real time tasks. Another component is represented by wireless sensors, which are built from a smaller microcontroller like an ESP8266 and Raspberry Pi Pico or a Raspberry Pi Pico W. This module, like the gateway, can have sensors connected to it via different interfaces, but it will be limited to I2C and UART. Usually, if the sensors are selected from a single manufacturer, maximum two I2C addresses can be used in most cases by a sensor, but there are exceptions, like LM75 that can have up to five addresses. Some manufacturers use the same I2C addresses as others, meaning that two different sensors might have the same I2C address. In the case of UART sensors, the limitation is made by the number of UART ports of the microcontroller. Figure 1 describes the general topology of the system.

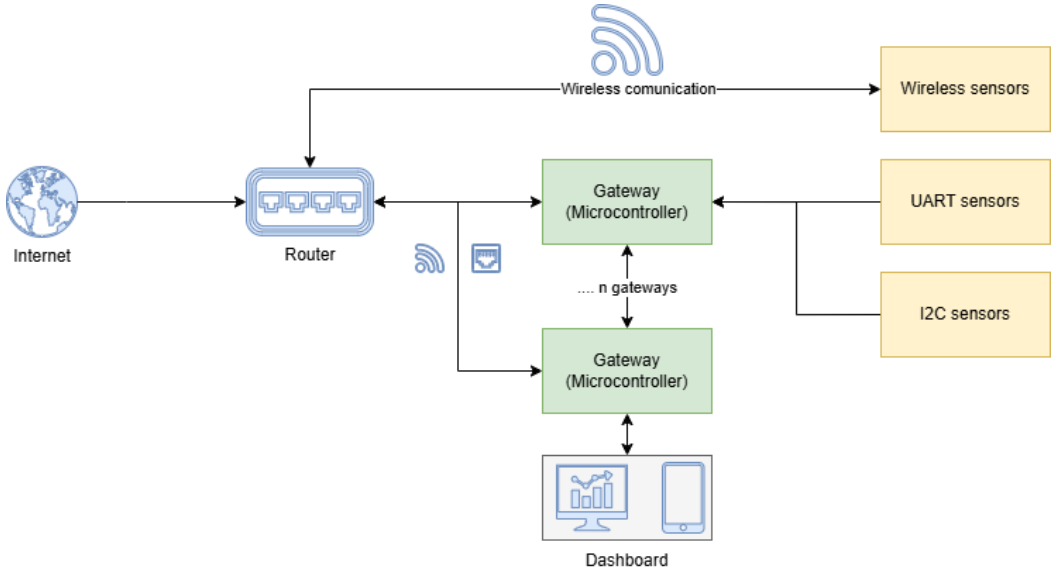

Figure 1. General topology of the proposed system (according to our own research)

The system contains an array of gateways and wireless sensors and one of the limitations is using an IP v4, because, theoretically, there is a possibility to connect 254 devices in the network, but the limitation is much lower in case of consumer grade routers. If the system is applied in an industrial environment or if the gateway provides a Wi-Fi network for the sensors, then the limit depends on the number of connections that be handled by the gateway, but, in general, the maximum number is 254. The limitation of using the IP v4 and 2.4 GHz Wi-Fi happens because most microcontrollers can use only this kind of Wi-Fi networks, and some are not yet compatible with IP v6, especially when cheap microcontrollers are used. Some consumer grade routers manufacturers recommend a maximum of 30-50 devices in a network, but the current hardware becomes more advanced, day by day.

From a software perspective, the system is composed of a server, user interface and a mobile client. The architecture is classic for this kind of project, as a client-server approach is used and no data is saved in any cloud solution. Every time a user asks to show data from a sensor, the request will reach first the main gateway, then the data will be requested to the gateway where the sensor is connected. The clients cannot request data directly from the wireless sensors, because the

connection mechanism does not allow it, for security purposes. The gateways can display the user interface that allows the user to see data and configure the modules, but some of the gateways can be configured to run without the user interface, especially if the system runs on systems with 512 MB of RAM or even less. The main gateway can be represented by any of the gateways in the system, but it is recommended to use a separate device which does not store data, to improve the system security and speed. This gateway is selected by the user when it first configures the system, using the mobile application, by scanning the Quick Response (QR) code from the interface. The system is in a continuous improvement process. One of its first versions was presented in (Vădan & Miclea, 2022), where a real home simulation with 9 gateways and 2 wireless sensors was used. The system was extended to other microcontrollers, newer wireless sensors, and the user interface was improved. General information about the system can be found in (Vădan & Miclea, 2023).

In the initial implementation, the system used two sensors to measure the air quality, SGP30 and CS811. Because SGP30 is very slow, if other sensors are connected, when the scan of the I2C bus occurs, only SGP30 is visible or only other sensors. It depends on case to case, due to clock stretching limitations. To bypass this problem, both Pi Pico I2C connections are required. It is recommended to chouse sensors that work at same speed or to connect slow sensors to a secondary I2C interface. In Figure 2, is shown a remote sensor with multiple sensors using both I2C interfaces of the Pi Pico.

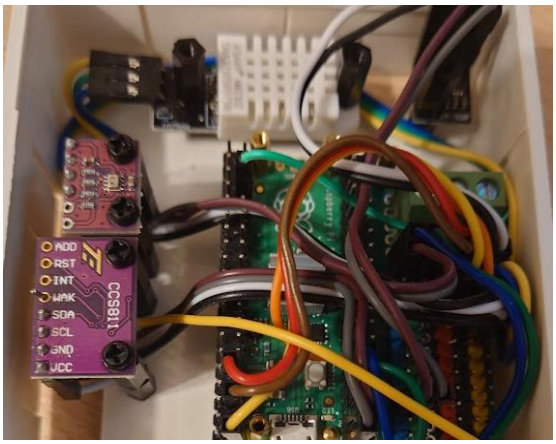

Figure 2. Remote sensors module using a Raspberry Pi Pico with a BMP280 and CCS811 sensor connected on I2C -1, SGP30 on I2C -2 and DHT22 (according to our own research)

4. Self-configuration of gateways and wireless sensors module with sensors

The proposed solution automatically recognizes the sensors attached to it. The sensors can be attached directly to GPIO ports using interfaces like I2C, SPI, UART, USB ports or by using other microcontrollers, wirelessly. The configuration of a such complex system can require advanced know-how and the automation of this process might help the end user to avoid learning how to configure the system, and in case of errors, when a system reboots, the damaged sensors are not initialized at all. To achieve a self-configuration mechanism, it is a must to self-test the configuration of the gateways. In the beginning, it is required to select the possible sensors that can be used in the system, a good example being the usage of sensors like BMP180, BMP280 or BMP388. After defining the sensors of the list, it is required to make the implementation of the scripts responsible for initializing and reading the data from a specific sensor. The I2C address is mentioned in the datasheet of the sensors and, in some cases, there are two possible addresses. For example, BMP280 can use both the ‘0x76’ and ‘0x77’ addresses, depending on where the sensor was bought from, but, in general, it should be ‘0x76’.

Both BMP388 and BMP180 use ‘0x77’ address, by default. In the system, it is impossible to use both BMP388 and BMP180 on the same I2C bus, because they have the same address, but BMP180 can be replaced with BMP280, if it is a must to use two similar sensors. In case it is

needed to have multiple sensors reading the same kind of data, it is strongly recommended to use sensors with similar specifications, from different manufacturers, for better accuracy. To configure the system with ease and to provide a simple solution for the end user, the sensors connected to I2C must be correctly detected and initialized. Most modern sensors have a register where it is stored a value that represents the chip identifier, a serial number or a unique identifier for a certain model of sensors. This register value, further called the "chip id", is a mechanism to differentiate sensor versions. When a new generation of sensors is made, the manufacturer changes or increments the value in this register. There are cases with older generation of sensors where this register is missing, but there are also cases where there are libraries, like the ones for BMP180 or BMP280, where reading this variable is not implemented. In some cases, the "chip id" register is not present at all, even in newer generations of sensors. In the ideal situation, like using BMP sensors, the scripts can be initialized and then the procedure of reading the register related to "chip id" can be executed. If the "chip id" property or similar identifier is missing from the library, but in the datasheet of the sensor it is available, it is recommended to create a function that can check this value. SHT3x sensor series do not provide any "chip id" register, so the version of sensor cannot be determined in a similar way with the BMP case. In general, using multiple sensors from the same manufacturer is not an issue, if there is a register where the versions of the sensor are stored, but in case of sensors like SHT3x, it is impossible to use multiple variations in the same system, on the same I2C bus. Both SHT3x and SHT4x sensors are using '0x44' address, but SHT4x sensors have a serial number register. OPT3001 also shares the same I2C address as SHT sensors and, in case of having both OPT3001 and SHT in the list of possible sensors, it is recommended to read first the "chip id" from OPT3001. If OPT3001 is not found, then try to check for a SHT4x and if the device id or serial number is not found, it can be assumed that SHT3x sensor is present. It is recommended to search first for sensors that are using a "chip id", in case there are multiple sensors using the same address. In the end, if the checks do not match any sensor, an assumption that a certain sensor without "chip id" is used can be made. In case the I2C address is in use, and there is no check for that address, the sensor attached will be just powered on. Some sensors like MAX30102 might require a signal to trigger the power on first.

To resume the above and achieve self-configuration of the sensors on a gateway module implementation, the following pseudo-code can be applied:

- First, a defined list with sensors and the corresponding I2C addresses is required:
    - The list must include only sensors that do not need a trigger signal to turn on and that are visible on I2C at system startup;
- The second step is to perform a scan of the I2C and obtain the list of used addresses;
- In the next phase, an iteration of the sensor addresses from the obtained list is required. There are 2 scenarios:
    - Scenario A: only one possible sensor use the current I2C address;
    - Scenario B: there are two or multiple sensors that can use the same I2C address:
        - First "chip id" reading is required,
        - If the found value corresponds to a certain sensor, then add that sensor in the list of found sensors and go to the next address,
        - If the "chip id" is not available, assume that a certain sensor is used,
        - Note: The assumption must be at the end after all the tries to identify the sensors that are possible to use the respective address occurred;
- After the list of all addresses is checked, a list of sensors should be obtained, and sensors initialization process can be started.

The diagram from Figure 3 represents initialization of sensors connected to a wireless sensor module in an ideal scenario when there are only two sensors using one I2C and one of the sensors has the chip identifier available. In case of an I2C address found in use and a sensor cannot be identified, the initialization will be skipped.

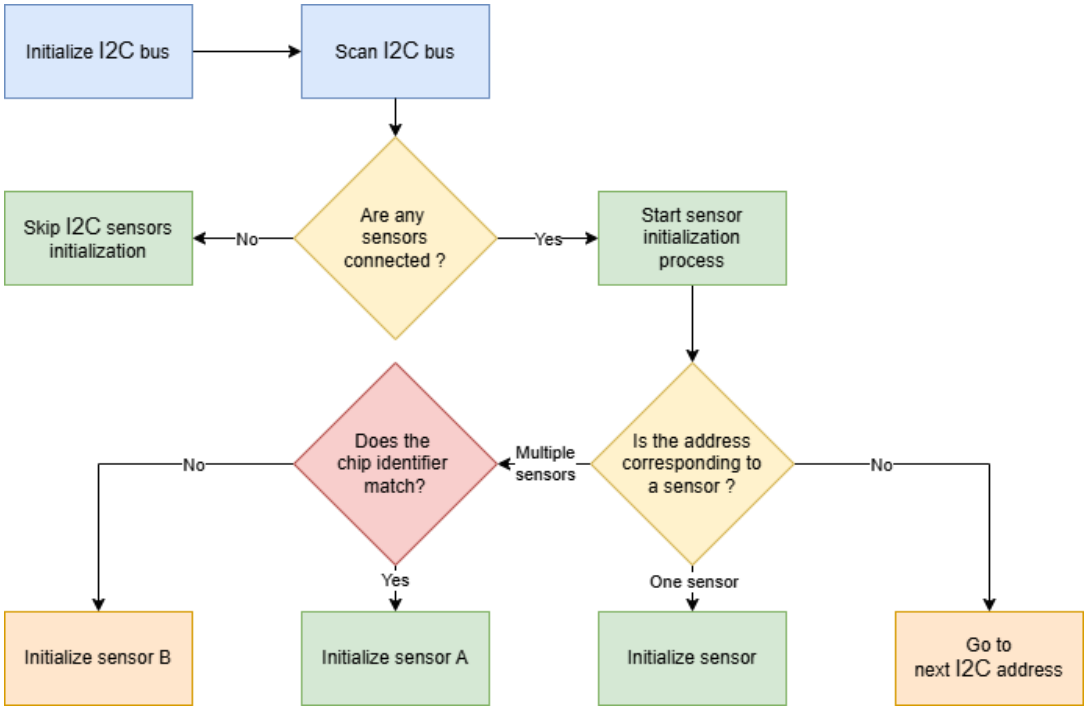

Figure 3. Handle basic initialization of sensors (according to our own research)

4.1. Self-configuration of the wireless module using Micro Python

In Circuit or Micro Python, because most of the microcontrollers have a limited amount of memory, the number of python modules that can be uploaded is small. From this perspective, it is recommended to define a precise purpose for each wireless sensor. An example can be a module for temperature and humidity reading. In this case, for example, there is a need to upload scripts for BMP180, BMP280, BMP388, BMP390, BME280, BME680, DHT20, AHT1x, AHT2x, SHT3x, SHT4x, HTU21D sensors. For a Pi Pico, it should not be a problem to upload all these modules, but some microcontrollers like ESP8266 come with 512KB of memory available for the actual code. In the case of low memory microcontrollers, it is a must to import only the specific python modules or remove some microcontroller boards from the package to save some space, especially if Adafruit modules are used. After uploading the required python modules, in the main coding file, the main logic must be implemented, which should contain some key features, like giving the user the ability to configure a Wi-Fi network, register a gateway, reading and sending data from sensors.

The wireless sensor should start with Wi-Fi set in an access point mode, until the user manually connects the wireless sensor to a Wi-Fi network, preferably via a Web interface. The Web interface should also contain a field where the user should be able to specify the hostname of the gateway. If this field is left empty, the sensor will not send data to any gateway, until the registration process is triggered by the automatically pairing mechanism. If the user completes the field, he must first enable pairing mode on the specified gateway. Only after a successful pairing, the gateway will accept data coming from the wireless microcontroller. In general, because communication is not accepted from any device, the system provides an extra layer of security. The fact that the microcontroller must provide certain data about it, to be paired successfully with the gateway, also increases the security. The data must be specific to that microcontroller, as in the case of a unique serial number. The gateway will try to build the same identifier from the data provided by the microcontroller, to validate that the identifier is correct and can be a trusted device. If the process is successful, the registration process starts, and the microcontroller is paired with the gateway. In case the user provides the IP address of the gateway, the pairing process takes less than 10 seconds, otherwise it is much slower, due to the network scanning process.

4.2. Self-configuring a gateway with sensors connected on UART or USB

When searching what sensors are better to use in the proposed system, sensors like MHZ19x series that require usage of UART bus were identified. To connect such sensors, it is recommended to use the available UART interface on GPIO ports or some "USB to UART" adapters. When using the GPIO ports, it is not possible to use multiple sensors on one interface and automatic sensors configuration is not possible. In this situation, there are other options, like a wireless module that was described before. The initialization of the sensors on UART is handled in the code running on the host microcontroller. This code must be responsible for initializing, reading, and printing data on the serial console, in a predefined interval of time or when the gateway requests it. The implementation of a such module is recommended to be done using Arduino, Circuit Python or Micro Python. Data printed on serial console must be formatted so that it can be easily parsed on the gateway. The string printed must contain the sensor name, the type of data it reads and the actual value. An example for DHT22 sensor can be: "DHT22: Temperature: 30℃, Humidity: 60%".

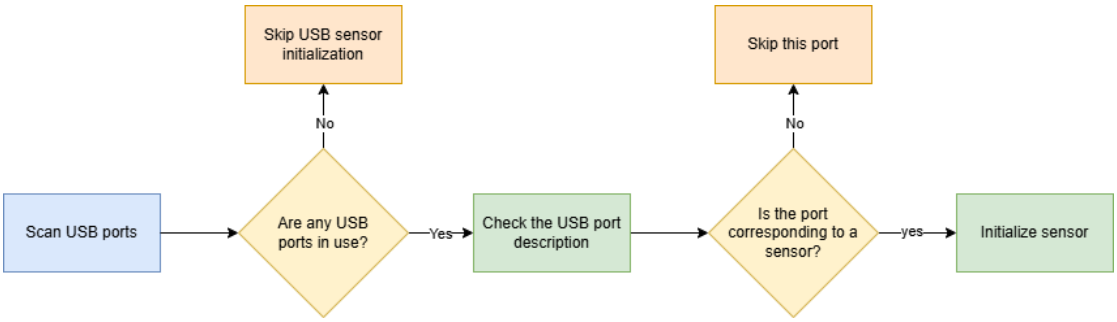

Figure 4. UART sensor initialization algorithm (according to our own research)

In Figure 4, the main steps required to initialize a sensor using an "USB to UART" adapter or connected to a host microcontroller, like a Pi Pico, are exemplified. The process is similar to the one shown as an example for I2C. First, a list of USB devices connected to the gateway must be defined. The next step is to read the ports description and then it is a must to search for some keywords, depending on what "USB to UART" adapter was used, or something like "pico", if a Pi Pico is used. When using this approach, instead of having an address like "0x00", the list contains the USB port in use. A dictionary with "COM_PORT" keys should be generated. The "COM_PORT" is represented by the USB port where the Pi Pico or the adapter is connected. On Windows machines, they are named "COM", but on Unix systems, these are named "ttyUSB". These keywords are followed by a number. The value can be set by the description. The usage of multiple "USB to UART" adapters is possible when, for each type of sensor, a different type of adapter is used and there is a way to identify a difference between them, for example, the device description is different. In the experiment proposed in this article, two MHZ19 sensors were used connected to the same type of adapters. The initialization succeeded, but when MHZ19x sensor was changed with a dust particle sensor, because the same type of USB to UART adapter was used, the initialization process resulted in an infinite loop, as the code was expecting a response after writing the initialization sequence on UART. From this perspective, the UART usage is very limited in comparison to I2C bus.

4.3. Self-configuration of the main gateway with other gateways and wireless sensors modules

There are different implementations for smart-home systems that use configuration files or databases to store sensors addresses or the configuration, but those are generated by a manual or predefined configuration of the system. To create a better experience for the end user or to have more flexibility in developing such systems, it is more convenient to automate the process of sensors initialization. In the proposed system, all the sensors are initiated at system boot and data starts being saved immediately. After the pairing of all gateways with wireless sensors, the system can work independently. Each gateway can represent a room or more, and, in case the gateway is

configured for multiple rooms, the user can select which sensors are used in each room. In most of the current available smart-home systems, the user must create a room and attach all the sensors to that room manually.

Most users want to access the system from remote locations, and, in this case a minimum-security system must be implemented. With the proposed system, the data are not sent to a cloud service, to access it over Internet. It is recommended to install a gateway without sensors, if possible, to have just one device exposed to Internet, to avoid data breach. This proxy device should block all incoming requests and allow only the ones that are implemented and have certain attributes. If the system is implemented using gateways with more than 1024 MB of RAM, the web interface and the API can run in a container within a Docker environment. On the router side, the user must create a port forwarder. The system must have a registration mechanism and only devices from a certain list are allowed to call the API and receive a response. By default, when the gateway is turned on for the first time, it should be in "pairing" mode. The user should be able to set the paring mode also from the web interface, when he desires to connect a new device in the system. The pairing process must consist in scanning the network using IP V4 and trying to call on each IP a certain webservice, like a "submodule", for example, with the required headers or parameters. If on a specific IP there is a web server running and it is a gateway, then the gateway should respond with a "device identifier" and some hostname details. In case the web server represents a wireless sensor, a unique identifier is also provided. The registration process is exemplified in Figure 5.

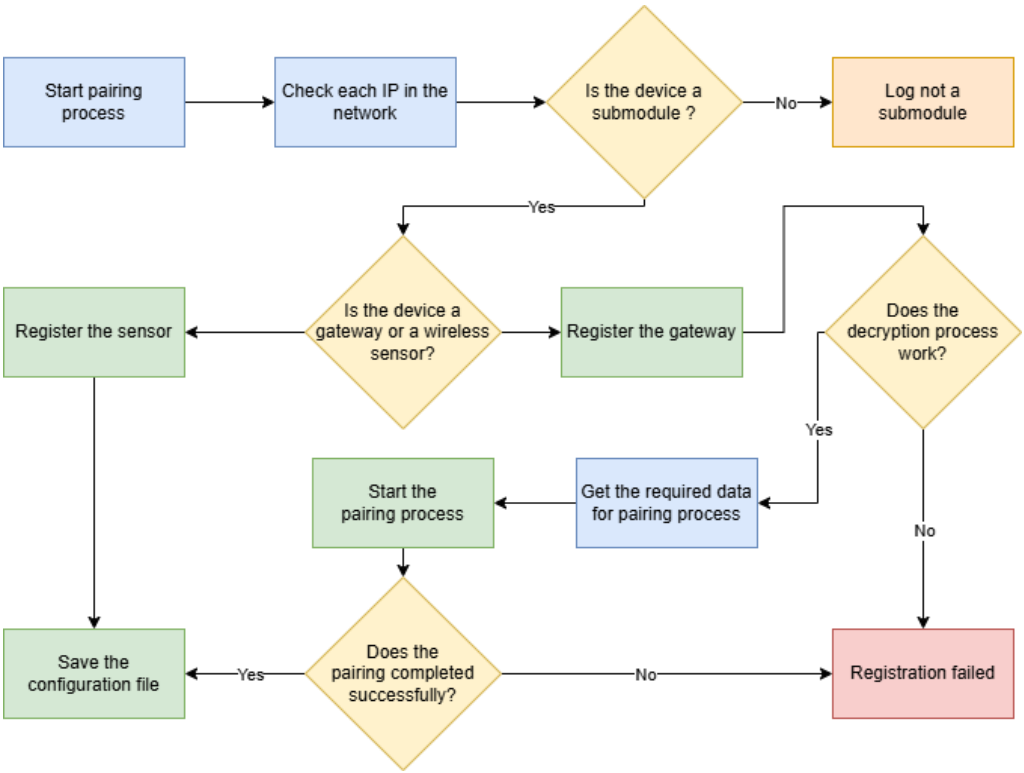

Figure 5. Registration process (according to our own research)

5. Experimental results of applying the methodology

The proposed system consists in a gateway and wireless sensors which can have attached a large number of I2C sensors. In an ideal situation, when there are not multiple sensors using the same I2C address, the solution should work flawlessly. All sensors must have the same speed mode to have a smoother operation. In most cases, I2C works in fast mode. If a microcontroller with multiple I2C ports is used, issues with sensors detection or reading data from them are avoided and a higher number of sensors can be used. Sensors that do not use fast mode can be connected to the secondary I2C bus, as it has been shown in Figure 2. Another aspect in choosing the sensors is their physical address on the I2C. It must not be the same address with other sensors, or, if it is the same,

it is a must to make sure that one of the sensors will work on a secondary address, because, in some rare cases, even if the sensor supports a secondary address, reading data from it does not work correctly, like the case of sensors using AHT library.

The automatic configuration of sensors over UART might work in cases where there is no need to write data first, only to read it. From the perspective of the present work and from the experience gained while working to create the proposed system, a better approach is to directly connect the UART sensors to a microcontroller like Pi Pico (W), instead of using USB to UART adapters or connect the sensors to the main gateway. Since there is a need to first write on UART and the sensor might not respond, the whole system can be blocked for a few seconds or until a sensor is connected, in some cases. This situation was encountered while trying to initialize the MHZ19 sensor. Because there was no error or response received on serial bus, the system got blocked, waiting for a response. Using microcontrollers like Pi Pico and Pi Pico W has a big advantage, because these microcontrollers have more UART ports than others. Another option when working with UART devices is the possibility to choose a simple ESP8266 or other ESP microcontroller instead, of a serial adapter, because the code required for the sensor initialization can be handled and a Wi-Fi connection can be established between this wireless sensor and a gateway with ease, by using Arduino or Micro Python.

Having a mechanism of pairing between remote microcontrollers and the gateway, using some unique identifiers and exchange of tokens, might improve system security. By using a gateway to gather and save data, there is no need to query the sensors each time, since the wireless microcontroller can be programmed to send data and to enter in a deep sleep cycle after that. This means that better power efficiency can be achieved and because, the gateway is not sending requests for data, the security level is much higher. The master gateway provides an additional layer of security when exposing the system to Internet.

In a home, sensors are required in different places, in order to control an irrigation system, monitor the air quality or to control different appliances. The cables length for I2C sensors should be less than 40 centimeters. In some cases, the sensors need to be put under the floor, and this limits the way the data are sent to the gateway. This situation is similar to one described in (Ferreira et al., 2019), but inside a building. In the case of a smart garden irrigation system, a method is described by Dickson & Amannah, (2023).

The solution presented should be easy to understand for most developers and easy to implement, since a client-server approach is the most used software architecture today. To make the solution compatible with multiple existing hardware and extend the capabilities, there are protocols like MQTT and IFTTT. For example, Xu et al. (2024) described a solution using MQTT technology. A client-server architecture was selected, due to the ease of implementation, and Python was selected as programing language, since is available across all the operating systems, for all microcontrollers. The solution in the present paper is close to a Fieldbus approach proposed by Zhang et al. (2024). The present work displays a master-slave configuration, based on the approach proposed by Chen et al. (2023). A similar conceptual architecture is described also in other papers (Ahmed et al., 2018; Stolojescu-Crisan, Crisan & Butunoi, 2021).

Vădan & Miclea (2022) presented a real-life example of the system described and a smart home with 8 gateways was simulated. In that case, the system used only the sensors related to temperature and ambient lightning. The type of sensors that were attached to each gateway was also described. In Vădan & Miclea (2023), the system is described in detail, the actual wireless modules using Pi Pico W and ESP8266 are shown, and the screenshots from the gateway and mobile client interfaces are displayed.

If the process of setting up a smart-home system for an entire home is compared with the one proposed in this paper, by using a Sonoff solution or other paid solutions, the input from the user proves to be minimal in the present solution. When using the present solution, the user must scan a QR code from the main gateway web interface, and trigger the scan for submodule process, which can take about 10 minutes. An example of the “Settings” page and the QR code can be seen in Figure 6.

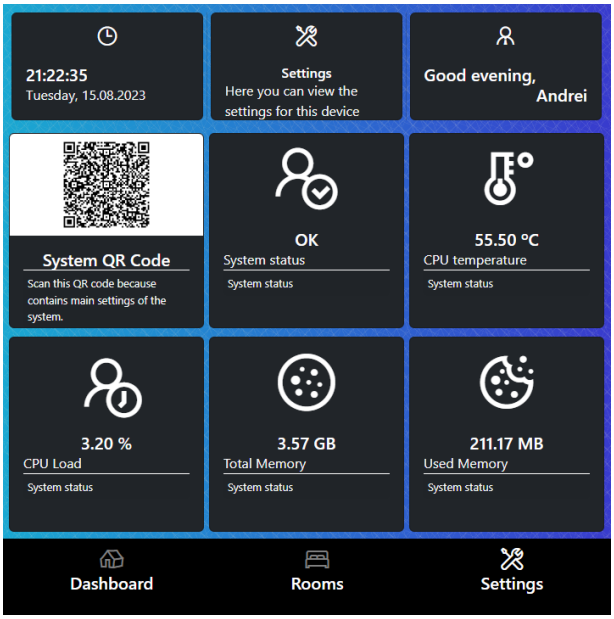

Figure 6. The web user interface from a module in "Settings" page (according to our own research)

After the scan is completed, the user can create a room and select the module desired for that room, then, the process is completed. In the application provided by Sonoff (eWeLink) or other solutions, the user must create first a house, then create a room. After the room is created, he must scan the sensor or device that he wants to attach to that room. In most cases, the Wi-Fi network from the phone is disconnected and the user has to manually scan the device. In some cases, the user must manually select the device from a list, which can also be outdated. There are mobile applications that are not updated by the users, or they no longer support the operating system on the mobile device. In some cases, the application running on the mobile phone used by the user is not compatible with a newer operating system, while, in other cases there is a probability of having the configuration saved only on that device. Usually, the configuration of the existing systems is more time consuming and, if the process does not work as intended, the user must search for the device in a list of devices and can easily miss the correct device. The use of an application like "Lg ThinQ" can be confusing, because there are 3 methods to manually pair a device. At present, when the user taps to manually add a device from TV category, an automatic search for devices process is started.

There is also a possibility that the user has products from different manufacturers, a case when he must install a specific application and only after performing some configurations, he can use the smart devices. Depending on the type of device, there are cases when the smart device does not offer support to use a generic application, especially when the device is new on the market. A similar method is employed when "Lg ThinQ" is configured, but, in this case, the user can add a device in the application without the need for setting up a room. However, the process of pairing a device is slow and it takes a few minutes, due to the user being required to process different inputs.

In comparison with other solutions, the proposed solution offers the user a certain level of comfort, by decreasing the time of configuring a house. Organizing devices becomes very easy and, of course, the manufacturer can deliver the same product with different sensors, without the need of creating special software or configuration files for each variation. When a sensor requires replacement, it can be easily done and the sensor can be replaced even with a newer model, the only restriction might be the software support. In a classic approach, with current solutions, the user needs to replace the entire device. The presented solution is an improvement over existing solutions, because it can use a larger number of sensors and if, a sensor is not working anymore, the system will continue to work and boot up. In a system where the sensors list is strict, it might enter in an error statement and the system cannot be further used by the end user, until the problem is fixed. No method to inform the user that a certain sensor is not working anymore has been proposed, but rather a potential system developer that can easily implement something similar.

6. Conclusions

This paper presents how a smart-home IoT system can be created by using a state-of-the-art network architecture, and popular and relatively cheap hardware, like Raspberry Pi, or Pi Pico and Pi Pico W or ESP8266. The system was successfully run using other devices, like Banana Pi and Nvidia Jetson Nano, which are more expensive. The main challenge was to create a system that can use different combinations of sensors connected to I2C bus and automatically configure them. The issue occurs when there are multiple sensors that might use the same I2C address, but this problem can be solved by reading the sensor id. When the sensor id is not available or reading it is not possible, an assumption that a certain sensor is used can be made.

The paper also presents a situation where the algorithm does not work, and the usage of sensors like AHT or DHT. The same approach can be applied to sensors using UART, but it is recommended to use a microcontroller instead of an UART to USB adapter. Another challenge was to configure remote sensors. When a microcontroller with a lower memory is used, the solution is to create a remote sensor with a specific purpose, like reading temperature, for example. The communication between gateways and wireless sensors or mobile clients happens with encrypted messages. Using a mechanism to automatically scan the Wi-Fi network, the configuration of the system is effortless for the end user, but still not fast enough, and it can be improved using technologies such as Bluetooth or Network Field Communication. Overall, because the system can automatically configure itself, communication happens only with encrypted messages. The system provides a better experience for the end user, and the concept can be successfully used in a situation where certain sensors are not available or for a quick replacement.

For simplification, it was limited to I2C and UART sensors, but for sure this can be extended to SPI or I3C or other types of interfaces. The introduction of this paper discussed about using microcontrollers like Banana Pi or Nvidia Jetson. The configuration is like the one of Raspberry Pi, the difference consisting in accessing the I2C interface and the higher price of the microcontroller. There are technologies like MQTT or IFTTT that can be included in the system, in order to support more types of sensors or devices. From a security perspective, including HTTP2.0 or HTTP 3.0 will increase the security. Storing encrypted data is also a must.

To sum up, the presented system provides a method for easy configuration of the entire system, using I2C sensors or UART. It can be used in different scenarios, but a smart-home system that can be easily configured or upgraded was chosen as example.

In future, the selection of microcontrollers can be done using Rădulescu & Neacșu (2023). System security can be improved significantly by applying artificial intelligence techniques presented by Jayaraman et al. (2023) as example.

REFERENCES

Ahmed, K., Blech J., Gregory, M. A. & Schmidt, H. H. W. (2018) Software Defined Networks in Industrial Automation. *Journal of Sensor and Actuator Networks*. 7(3), 33. doi:10.3390/jsan7030033.

Campbell, S. (2021a) Basics of the I2C communication protocol. *Circuit Basics*. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/ [Accessed 29th December 2022].

Campbell, S. (2021b) Basics of UART communication. *Circuit Basics*. https://www.circuitbasics.com/basics-uart-communication/ [Accessed 29th December 2022].

Chen, L. Zheng, J., Fan, D. & Chen, N. (2023) Research on the High Precision Synchronous Control Method of the Fieldbus Control System. *Machines*. 11(1), 98. doi:10.3390/machines11010098.

Davidovic, B. & Labus, A. (2016) A smart home system based on sensor technology. *Facta Universitatis.* 29(3), 451–460. doi:10.2298/fuee1603451d.

Dickson, M. & Amannah, C. (2023) Augmented IoT Model for Smart Agriculture and Farm Irrigation Water Conservation. *International Journal of Intelligence Science*. 13(4), 131-163. doi:10.4236/ijis.2023.134007.

Ferreira, C., Peixoto, V.F., G. de Brito, J.A., Monteiro, A.F., Silva de Assis, L. & da Rocha Henriques, F. (2019) UnderApp: A System for Remote Monitoring of Landslides Based on Wireless Underground Sensor Networks. *Companion Proceedings of the 25th Brazilian Symposium on Multimedia and Web*. pp. 79-82. doi:10.5753/webmedia_estendido.2019.8142.

Froiz-Míguez, I., Fernández-Caramés, T. M., Fraga-Lamas, P. & Castedo, L. (2018) Design, Implementation and Practical Evaluation of an IoT Home Automation System for Fog Computing Applications Based on MQTT and ZigBee-WiFi Sensor Nodes. *Sensors*. 18(8), 2660. doi:10.3390/s18082660.

Jayaraman, B., Thanga Nadar Thanga Thai, M., Anand, A. & Anandan, K.R. (2023) Detecting malicious IoT traffic using machine learning techniques. *Romanian Journal of Information Technology and Automatic Control [Revista Română de Informatică și Automatică]*. 33 (4), 47-58. doi:10.33436/v33i4y202304.

Martin, J. (2021) How many devices can connect to a router at the same time? *Tech Advisor*. https://www.techadvisor.com/article/742761/how-many-devices-can-connect-to-a-router-at-the-same-time.html [Accessed 20th December 2021].

Ożadowicz, A. (2024) Generic IoT for Smart Buildings and Field-Level Automation—Challenges, Threats, Approaches, and Solutions. *Computers*. 13(2), 45. doi:10.3390/computers13020045.

Rădulescu, C. Z. & Neacșu, D. M. (2023) Selection in IoT type systems based on multi criteria methods. *Romanian Journal of Information Technology and Automatic Control [Revista Română de Informatică și Automatică]*. 33(3), 113-128. doi:10.33436/v33i3y202309.

Stolojescu-Crisan, C., Crisan, C. & Butunoi, B.-P. (2021) An IoT-Based Smart Home Automation System. *Sensors*. 21(11), 3784. doi:10.3390/s21113784.

Vădan, A.-M. & Miclea, L.C. (2022) Detect Data Deviation for Temperature and Ambient Light Sensors, and Create a Simple Calibration Method. In: *2022 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), 19-21 May 2022, Cluj-Napoca, Romania*. IEEE. pp. 1-6. doi:10.1109/aqtr55203.2022.9801986.

Vădan, A.-M. & Miclea, L.-C. (2023) Software Testing Techniques for Improving the Quality of Smart-Home IoT Systems. *Electronics*. 12(6), 1337. doi:10.3390/electronics12061337.

Xu, H., Seng, K.P., Smith, J. & Ang, L.M. (2024) Multi-Level Split Federated Learning for Large-Scale AIoT System Based on Smart Cities. *Future Internet*. 16(3), 82. doi:10.3390/fi16030082.

Zhang, L., Hu, X. & Wu, B. (2024) Applications and Challenges of Fieldbus Technology in Intelligent Manufacturing Systems. *Applied Mathematics and Nonlinear Sciences*. 9 (1), 1-17. doi:10.2478/amns-2024-0900.

Andrei-Mihai VĂDAN started his professional activity in 2013, with developing applications for mobile phones and web development, working on more than 30 projects, but working on the first translation software in the world was noticeable, since the project is almost the same age as him. He was certified as a tester by ISTQB, in 2016, while between 2017 and 2019 he

followed and successfully finished the master's degree courses at Technical University of Cluj-Napoca. He started the PhD studies in the same year. Currently, he is the technical leader of the automation team at Porsche Engineering Romania, in the Infotainment department.He presented his automation framework concepts at SEETB Conference, in 2019 and 2022, where it was selected as one of 20 speakers. His fields of interests are focusing on automation testing, IoT, smart-home systems, embedded systems, automotive industry, and operating systems.

Andrei-Mihai VĂDAN și-a început cariera profesională în 2013, dezvoltând aplicații mobile si web și contribuind la peste 30 de proiecte, printre care se numără primul software de traducere din lume. În 2016, a obținut certificarea ISTQB, iar între 2017 și 2019 a urmat programul de master la Universitatea Tehnică din Cluj-Napoca pe care l-a finalizat cu succes. În același an, a început studiile doctorale în cadrul aceleiași universități. În prezent, este coordonator tehnic al echipei de testare automată la Porsche Engineering Romania. De asemenea, a fost selectat printre primii 20 de speakeri la conferințele SEETB din 2019 și 2022. Principalele sale interese de cercetare includ testarea automată, Internetul Lucrurilor (IoT), casele inteligente, sistemele embedded, domeniul automotive și sistemele de operare.

Liviu-Cristian MICLEA graduated Faculty of Automation and Computer Science of the Technical University of Cluj-Napoca in 1984 and earned a PhD in Automatic Systems in 1995. From 1995 until 2004 he was Lecturer and Associate Professor, since 2004, he has been a full professor, between 2003-2011 he served as head of the Automation Department, and between 2012-2024 as dean of the Faculty of Automation and Computer Science. He is now Vice President of the Senate, is the author or coauthor of 18 books, 5 book chapters, 45 research projects and more than 280 scientific publications. His research interests include dependability, cyber-physical-systems, agent systems. Dr. Miclea is corresponding member of Romanian Academy of Technical Sciences, senior member of IEEE, liaison for Romania of IEEE TTTC, member of IEEE Computer Society, of IFAC Romania and of SRAIT. In 2008 and 2015, he received a Meritorious Service Award from the IEEE Computer Society, for his fructuous IEEE activity. He is regular the general chairman of the biennial IEEE-CS-TTTC-AQTR conference. He was the general chair of the IEEE-ETS 2015 and DDECS 2019 symposiums.

Liviu-Cristian MICLEA a absolvit Facultatea de Automatică și Informatică a Universității Tehnice din Cluj-Napoca în 1984 și a obținut titlul de doctor în Sisteme Automate în 1995. Din 1995 până în 2004 a fost sef de lucrari și conferențiar universitar, din 2004, este profesor titular, între 2003-2011 a ocupat funcția de șef al Departamentului de Automatică, iar între 2012-2024 ca decan al Facultății de Automatică și Calculatoare. Acum este vicepreședinte al Senatului, autor sau coautor a 18 cărți, 5 capitole de carte, 45 de proiecte de cercetare și peste 280 de publicații științifice. Domeniul său de cercetare include dependabilitatea, sistemele cyber-fizice, sistemele de agenți. Dr. Miclea este membru corespondent al Academiei Române de Științe Tehnice, membru senior al IEEE, responsabil pentru România al IEEE TTTC, membru al IEEE Computer Society, al IFAC România și al SRAIT. În 2008 și 2015, a primit un premiu pentru servicii merituoase din partea IEEE Computer Society, pentru activitatea sa fructuoasă în cadrul IEEE. Este președintele general al conferinței bienale IEEE-CS-TTTC-AQTR. A fost președintele general al simpozioanelor IEEE-ETS 2015 și DDECS 2019.