Remotely Operated Robot with Live Camera Feed

Stan Ovidiu and Liviu Miclea

***Abstract*—In recent years robots have begun to become an active presence in everyday life in various forms. Helpful robots that offer help and support to people with special needs or social robots that are able to interact with customers in stores. The basic challenges that lie in improving their utilization and development are that they do not interact with each other and cannot use the information from one to the other. This paper aims to present a method of how to create a low-cost Android operated robot in order to validate the proposed architecture for an Edge-Computing Cloud in which the robots can be interconnected in a digital society.**

***Index Terms*—Robotics, robotics-cloud, edge computing.**

I. Introduction

For years robots have been slowly making their way into every aspect of our lives from huge scale industrial processes to every day around the house appliances and small gadgets. The unprecedented rise of the technology industry has given us the opportunity to make our day to day lives much easier. When thinking of the term robot, people associated them with small scale electrical humanoids that didn't serve a purpose or have an impact on the lives of people besides entertainment. Over the years this changed, and the public opinion changed slowly from useless contraptions to extremely helpful machines. They started doing the work that humans used to do ex-cept they do it faster, better and they don't get tired or demand raises. After workers and companies understood that this is the future they embraced it and started improving robotics technology to make them more effective, cheap and reliable.

One of the main reasons that robots have gained all this popularity is in part that they can help with small tasks around the house for example automated vacuum cleaners, lights windows and so on, they make lives easier and public perception slowly changed for the better.

The problem is that the public doesn't realize how much more robots do for us; they do the work no human wants or can do. They handle materials and substances that are highly dangerous. Nowadays robots are each designed specifically for one task, their design and characteristics have been highly optimized over the years. One of the best examples of this phenomenon are robotic arms, which are used as all robots from day to day tasks as professional photography to handling highly volatile substances such as uranium or other radioactive substances without the risk of radiation poisoning.

Manuscript received October 18, 2018; revised December 23, 2018. This work was partially supported by the by the ROBIN (PN-III-P1-1.2-PCCDI-2017- 0734) project.

Stan Ovidiu and Liviu Miclea are with the Technical University of Cluj Napoca, Faculty of Automation and Computer Science, Department of Automation, Cluj Napoca, Romania (e-mail: ovidiu.stan@aut.utcluj.ro).

Robotic arms have come a long way since their debut in 1954 [1], they revolutionized the manufacturing industry, conceived and then patented in 1961.The first instance had three degrees of freedom, a hunk of metal that weighed a staggering 1200 kilogram and was put to use in factory automation in New Jersey.

Nowadays robots are extremely varied and have a lot of purposes as stated earlier and due to the development of the technology's robots can also collaborate, request additional information from existing data warehouses in Cloud environments [3]. Our goal is creating software and services modules for the use of robots in an interconnected digital society, enabling companies to develop complex, intelligent and performing products and services for these users as well as society as a whole. Such robot systems are based on sensors, which retrieve data and analyze, and higher-level information is derived from which rules systems are built [2]. These data processing and processing systems (of large size, or generated in high speed streams - e.g., a camera mounted on a stand-alone machine captures several frames per second) require in-creased processing and storage capabilities, which is why many companies prefer the use, where possible, of a Cloud support environment. Data is captured by such sensors, transferred through various wireless or wireless communications technologies to the Cloud, where data processing and retrieval are transmitted back to intelligent control mechanisms [4].

II. Background

In order to create a complex Cloud environment in which all the dependency based and machine learning algorithm firstly we design and proposed an Edge Cloud system through which the robots can communicate and collaborate. The overall architecture is presented in Fig. 1.

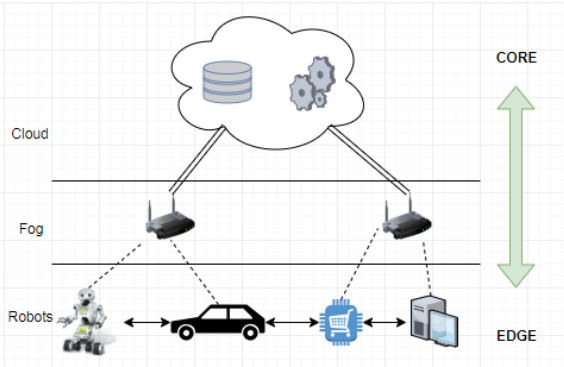

Fig. 1. Proposed cloud architecture.

Robots can be very different, from humanoids to intelligent self-driving cars or just IoT systems that collect and processes locally information from sensors. Each of the robots can

DOI: 10.7763/IJMO.2019.V9.682

communicate with each other if they are close enough or through the access points (AP). These robots generate all sorts of data such as equipped sensors data (light, temperature, gas, etc.), localization information, multimedia data, engine data, etc. Basically, the robots represent the Edge level of our architecture. In cloud, all the information's and data generated by the robots are collected and stored but in the same time the cloud is also use in order to process the information's.

III. Robot Implementation

In order to validate our concept, firstly we design a basic Android controlled robot with video live stream. The overall system is presented in Fig. 2. The Raspberry Pi camera provides a continuous video stream to the local wireless network, being hosted on port 8000 [7]. This address is used by the Android device to provide the operator with direct camera feed of the robot's environment. The Android device can set a desired speed for the robot, and using the on-screen buttons is able to transmit to the robot system its orientation parameters.

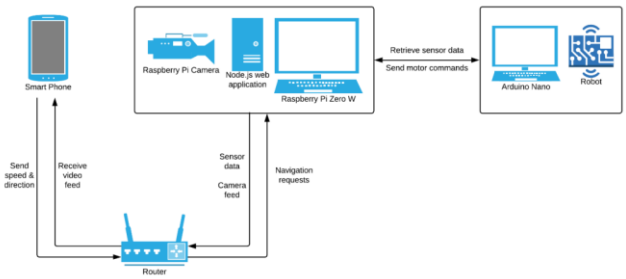

Fig. 2. System diagram.

On the Raspberry Pi Zero W housed by the robot runs a Node.js express server application which handles communication between the control device through a RESTful API service and the Arduino Nano board through I2C bus serial communication. Through the received input commands from the Android smartphone, the server interprets and handles the requests accordingly by constantly feeding direction and speed data to the embedded device. Navigation parameters of the robot are updated continuously, its direction data being fed as long as the operator holds down the button for a certain direction, while motor speed is adjusted and tuned by a PID controller relative to the imposed set point from the device.

A. Hardware Design

The robot, as show in Fig. 3, is composed of an Arduino Nano ATMEGA328P board for interfacing with the hardware components, a L298N Dual H-Bridge for motor control, 2 rotary encoders with infrared speed sensors to facilitate data acquisition for motor tuning, 2 AC5712 analog current sensors, a Raspberry Pi Zero W for I2C communication, and a camera module attached to the Raspberry Pi. DSN2596 and LM7805 are voltage regulators for the 2 connected microcontrollers.

A LI-PO battery feeds supply voltage to the H-bridge; the supply can be cut at any given moment through a switch placed in the center of the robot. Because the 12V jumper is in place, the H-bridge feeds an output of 5V to power up the Raspberry Pi and the Arduino Nano board.

The robot does not make use of stepper motors, and as such the jumpers on ENA (Motor A) and ENB (Motor B) are removed. Instead, the pins are connected to the Arduino to facilitate PWM (Pulse-width modulation) output, enabling DC motor speed control.

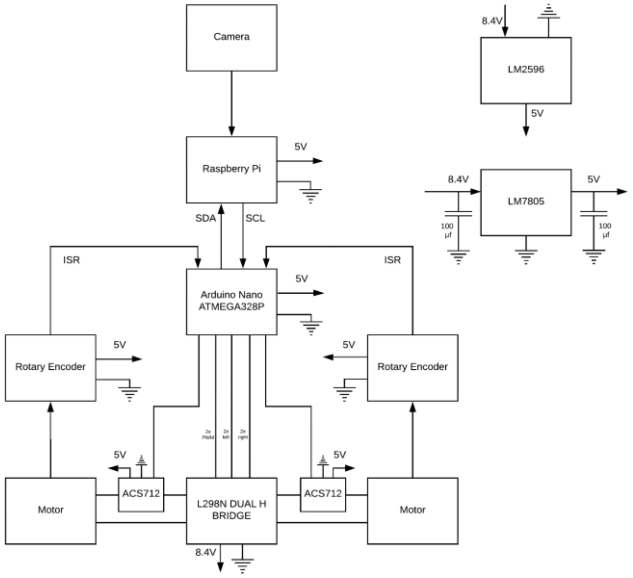

Fig. 3. Block diagram of the robot.

The L298N Dual H-Bridge is composed of four switching elements which control each individual motor through pairs of HIGH and LOW signals.

By supplying the appropriate HIGH and LOW signal to the channel of a DC motor we can dictate the direction it will turn to by reversing the polarity of the DC current applied to it. For instance, HIGH to D4 and LOW to D5 will cause the left motor to reverse, while LOW to D4 and HIGH to D5 will cause the left motor to turn forward.

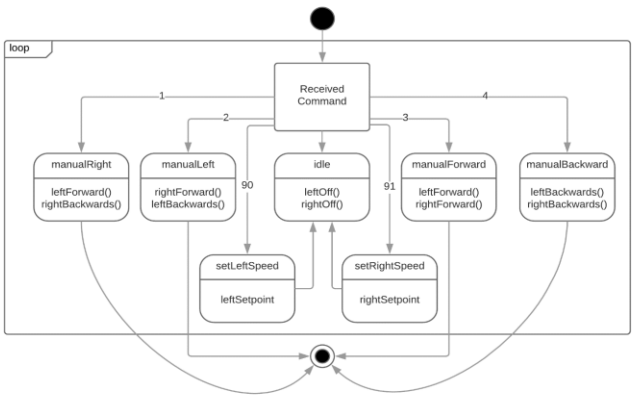

Fig. 4. State flow diagram of the system.

The corresponding control for each individual motor has been mapped and an appropriate finite state machine flow [6] implemented to achieve precise navigation control at any given time (Fig. 4).

To further improve the performance of the device and allow precise navigational control we employ the use of the

two PWM outputs. Taking the initial system into account, the DC motors employed only two possible speeds as a result of continuous voltage supply. The PWM comes into play by supplying the motor with voltage in a series of pulses, with wider pulses being associated with a faster turn speed, and narrower pulses with slower turn speeds.

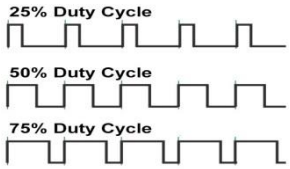

Fig. 5. PWM example at different duty cycles.

To further improve the reliability of our system and accentuate its scalability for future design and implementation, the received speed from the control system and subsequent PWM signal output payload is run through a PID structure [[8]] and continuously computed to provide an appropriate speed with respect to the imposed set point.

For acquiring the necessary motor data for controller tuning we employ the use of two rotary encoders that feature an infrared speed sensor. To this end, a python script was written on the Raspberry Pi device to interact with the robot system.

For accurateness of the acquired data we use three different speed levels: low (20), medium (50), and high speed (90). In each of the cases, the script dictates to the robot system the imposed set point and a direction to navigate. The robot interprets the set point signal, computes the current speed with respect to the rotary encoder's specifications, and transmits it further to the DC motor as a PWM signal. The slave device then responds to the Raspberry Pi with the desired speed readings.

This process is repeated three times for the varying speed values imposed. A data sampling of 150 readings per individual motor was deemed sufficient for system identification. The device was tested on different surfaces for more accurate data readings and for the purpose of replicating real world scenarios that could impede movement such as rough or slippery terrain.

The Arduino Nano ATMEGA328P is the core of the robot and the main microcontroller interface that manages the rest of the system.

The Raspberry Pi Zero W is installed parallel to the Arduino Nano board and facilitates I2C bus communication for retrieving sensor data and sending command signals to the board. A Node.JS server runs locally on the Raspberry Pi, listening for requests from the Android device and redirecting them to the robot.

The camera module transmits a live camera feed on a local hosted network, allowing the Android device to view the environment from the perspective of the robot, albeit with a delay.

The DSN2596 is a voltage step-down transformer meant to control the output voltage to the Raspberry Pi Zero W. An LM7805 positive voltage regulator is used to deliver fixed voltage to the system, and features internal current limitation and thermal shutdown from overheating, overall protecting the system from dangerous hazards.

The L298N Dual H-Bridge denotes the motor controller of the system, complete with 2 ACS712 linear current sensors (1 for each motor) for monitoring individual motor currents.

The rotary encoders were used for system identification in order to determine individual motor speed and tune them to fit our needs. Using the tuned data, the Arduino board is able to manage each motor's speed depending on a set point imposed by the Android device.

B. Software Design

The core of the application is based on the Node.js runtime and the server runs on the principle of asynchronous events. This facilitates a clean structure for triggering specific events that dictate the state flow of the system. The HTTP server implementation is deal for listening on specific endpoints for incoming HTTP requests as a way to transmit complex payloads.

The server manages data transmission to the robot system through the embedded modules. Asynchronous operations are exposed on the local network through specific routes. The server makes use of standard data parsing for maintaining a steady flow. Each call employs callback functionality when delivering successful process result payload, or handling error messages in the case of system failures.

The App.js file lies at the core of the server implementation and constitutes the integrated modules. A detailed structure of the server architecture and the relationship between the different Node.js modules are presented in Fig. 6.

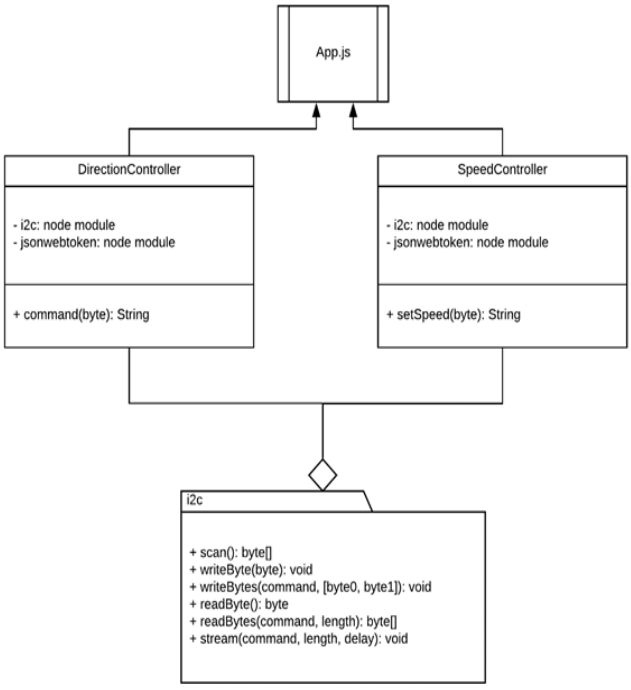

Fig. 6. Node.js server diagram.

1) I2C bus protocol

I2C is a synchronous serial computer bus used for communication between the Raspberry Pi Zero W and the embedded Arduino Nano ATMEGA328P microcontroller. It is a multi-master, multi-slave communication protocol widely used for short-distance communication between an integrated

circuit and a microcontroller. It facilitates a bus speed of up to 100 kb/s in standard mode and up to 400 kb/s in fast mode.

In the project's context, the Raspberry Pi Zero W acts as the master device, requesting data from the Arduino board (or the slave device), expecting a response. It is a relatively simple communication protocol feasible for our needs in retrieving sensor data and delivering adequate speed command signals from the control device.

2) HTTP through RESTful API services

For data transmission on a local area network, HTTP has proven to be reliable, flexible, and accessible. REST (Representational State Transfer) is an architectural style that aims to impose upon the protocol a set of constraints to facilitate easier communication between systems. In essence, server and application design can change in a controlled environment without repercussions in the operating mode of the system components. User interface is separated from the data layer, ensuring flexibility across platforms and managing increased scalability server side.

It is a form of stateless protocol implementation similar to HTTP itself, and thus RESTful APIs integrate seamlessly. Because of its nature, communication relies on an exchange of resources and do not rely on interfacing. This difference ensures REST applications achieve high performance and scalability, as the entire client-server ecosystem is not affected by constant updating and reusing of resources. Clients communicate with servers through requests in which they can pass extra information through headers or pass data through an optional message body. The four basic request types are: GET (retrieve a specific resource), POST (create a new resource), PUT (update a resource), and DELETE (remove a resource). It is similar in practice to CRUD operations specific to applications used for database management. Depending on the request path and the data transmitted, the server sends back a response with the appropriate status code along with data, should the requirements dictate.

For its excellent performance and scalability, RESTful API services facilitate an optimized client-server communication through resource payload exchange.

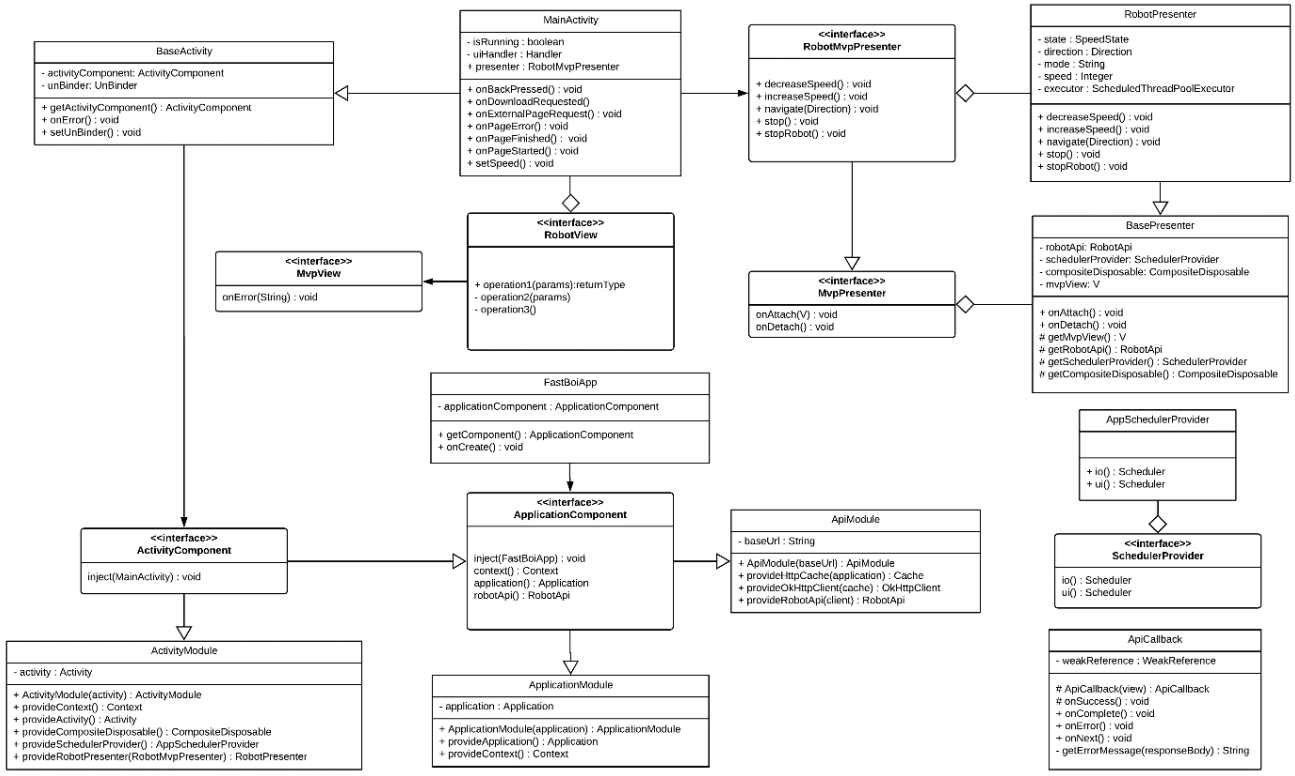

Fig. 7. Android application diagram. The advantage of dagger in android applications is the generation of required code without the use of reflection, which is a slow process in general and especially memory heavy for mobile devices. It provides compile-time assurance of necessary resource allocation.

3) Android application architecture

Due to the intricate interaction between the lifecycle of Android applications and the available resources, decoupling of standard Java/Kotlin language from the Android specific classes is required to facilitate stability. This is further made possible through a dependency injection framework known as Dagger 2.

The Dagger framework is built upon code generation and annotation processing for providing the necessary dependencies and injecting resources where it's needed: @Module and @Provides through which Dagger defines the classes and methods that provide dependencies; @Component, which are higher level interfaces that can include multiple modules, or even other components. They

are responsible for exposing the injection target and provide the dependencies declared by the modules; @Inject is used to request dependencies from the appropriate component.

Injected components are provided all the methods declared in the base modules. Dagger handles automatic implementation of the injected component at compile-time and detects potential errors, warning the developer of unresolved or missing dependencies.

For handling HTTP communication to the Node.js server we employ the use of Retrofit, a type-safe HTTP client that easily bridges API to Java/Kotlin [9]. It provides additional functionalities like caching, request interceptors, data conversion factories, custom request headers and file downloading. Its implementation is similar to asynchronous events on the Node.js server where data transmission is made possible through exposed endpoints, and as such is ideal for our project.

To facilitate abstractization of the underlying business logic we employ a Model-View-Presenter architecture to separate network calls from the View layer (in the case of Android applications, views are generally associated with Activities and Fragments). The Model layer is responsible for handling data flow, caching, and API requests. The Presenter is responsible for querying the model and updating the view, and thus decouples operations from the Android code base. This facilitates separation of the presenter from the Android framework and allows the developer to run non-instrumented tests.

IV. Conclusion

This paper proposes a method on how to a remote operated machine using live feed camera in order to demonstrate the functionality of the Cloud-Edge architecture for robots. The Figure 8 shows how the robot looks like and also the Android application used in order to control it.

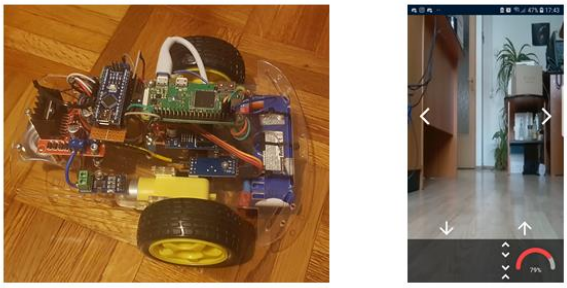

Fig. 8. The results of the above analyzes. In the left is the developed robot and in the right is the Android application with live camera feed and the controls (direction and speed).

The multimedia data transmitted by the robot is taken in the fog by AP and together with the information regarding the PWN and the status of the current used are forwarded to the cloud where at the moment they are just stored without being processed.

Acknowledgment

The research presented in this paper is supported by the ROBIN (PN-III-P1-1.2-PCCDI-2017- 0734) project..

References

[1] M. E. Moran, "Evolution of robotic arms," *Journal of Robotic Surgery*, July 2007, vol. 1, no. 2, pp. 103–111, 2007。

[2] K. Ben, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, 2015, pp. 398-409.

[3] K. Anis and E. Shakshuki, "Robots and sensor clouds," Springer International Publishing, 2016.

[4] E. Gabriele, S. Rosa, and A. Toma, "fly4 smart city: A cloud robotics service for smart city applications," *Journal of Ambient Intelligence and Smart Environments*, vol. 8, no. 3, 2016, pp. 347-358.

[5] H. Poor, *An Introduction to Signal Detection and Estimation*; New York: Springer-Verlag, 1985, ch. 4.

[6] S.-J. Wang and M.-D. Horng, "State assignment of finite state machines for low power applications," *Electronics Letters*, vol. 32, no. 25, 1996.

[7] R. Ikhankar, S. Ulabhaje, M. Dhadwe, V. Kuthe, and S. Balpande, "Pibot: The raspberry pi controlled multi-environment robot for surveillance & live streaming," in *Porc. 2015 International Conference on Industrial Instrumentation and Control (ICIC)*, Pune, India, 2015.

[8] Q. J. S. Li, "Study on pid parameters tuning method based on matlab/simulink," in *Porc. 2011 IEEE 3rd International Conference on Communication Software and Networks*, Xi'an, China, 2011.

[9] D. Zhang, S. Lin, Y. Fu, and S. Huang, "The communication system between web application host computers and embedded systems based on node.JS," in *Porc. 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, Shanghai, China, 2017.

Stan O. is a lecturer in the Automation Department at the Technical University of Cluj-Napoca. His research interests include medical informatics, semantic interoperability, information management in the age of the Internet, dependability and fault-tolerant systems. Stan received a PhD in systems engineering from the Technical University of Cluj-Napoca. He is a member of IEEE.

Miclea L. is a full professor the Automation Department at the Technical University of Cluj-Napoca. He is also the dean of the same faculty. He is the author or co-author of 17 books, 40 research works and more than 180 scientific publications. His research interests include: dependability, cyber-physical-systems, agent systems. Miclea is a Senior member of IEEE and is regular the general chairman of the bi annual IEEE-CS-TTTC-AQTR conference.