Journal Computer Technology 8[20]1-11: e40820111
Article

A proposal for an IoT operating system for plug-n-play wireless sensors

Propuesta de un sistema operativo para IoT para sensores inalámbricos inteligentes tipo plug-n-play

Moreno, Paul*[a], Baltazar, Rosario[b] and Casillas, Miguel Ángel[c]

[a] Instituto Tecnológico de León • LTE-5105-2024 • 0009-0001-1157-120 • 1265027
[b] Instituto Tecnológico de León • V-6474-2019 • 0000-0002-8847-8732 • 30501
[c] Instituto Tecnológico de León • LUY-1930-2024 • 0000-0003-1758-4092 • 79155

CONAHCYT classification:

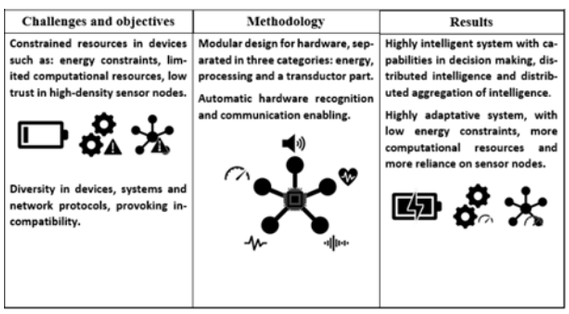
Area: Engineering
Field: Technological sciences
Discipline: Computer technology
Subdiscipline: Artificial intelligence

https://doi.org/10.35429/JCT.2024.8.20.4.11
History of the article:
Received: January 10, 2024
Accepted: December 30, 2024

* ✉ [rosario.baltazar@leon.tecnm.mx]

Abstract

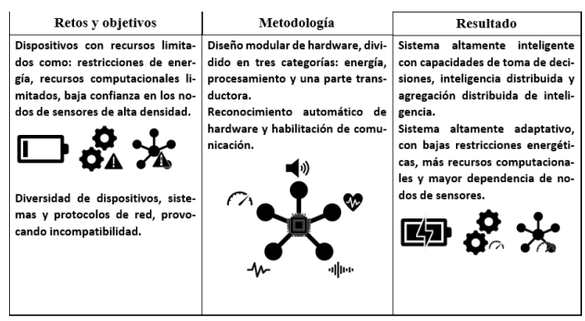
The proposal is focused on the development of an operating system that reduces the gap in the needed technological knowledge for the installation of an intelligent environment. The aim is to develop an operating system that is sufficiently advanced to autonomously manage the inclusion of new devices, and that at hardware level becomes largely versatile to integrate new devices and components into the environment. The proposal is based on modular hardware composed on three main elements: a brain, a module (a transductor) and a power supply. Consequently, the software must be able to recognize the installed hardware and subsequently manage communication with other devices with minimal human intervention, being helped by algorithms and fuzzy logic. Therefore, the contribution focuses on the creation of ubiquitous and pervasive systems, where the system manages itself and benefit.



Ubiquitous environments, modular system, autonomous elements

Resumen

La propuesta se centra en el desarrollo de un sistema operativo que reduzca la brecha en el conocimiento tecnológico necesario para la instalación de un entorno inteligente. Se pretende desarrollar un sistema operativo lo suficientemente avanzado como para gestionar de forma autónoma la inclusión de nuevos dispositivos, y que a nivel de hardware sea en gran medida versátil para integrar nuevos dispositivos y componentes en el entorno. La propuesta se basa en hardware modular compuesto por tres elementos principales: un cerebro, un módulo (transductor) y una fuente de alimentación. En consecuencia, el software debe ser capaz de reconocer el hardware instalado y gestionar la comunicación con otros dispositivos con mínima intervención humana, ayudándose de algoritmos. Por tanto, la contribución se centra en la creación de sistemas ubicuos y pervasivos, donde el sistema se auto gestione en beneficio de los humanos.



Entornos ubicuos, Sistema modular, Elementos autónomos

Citación: Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.

Introduction

In recent years the Internet *of Things*(IoT) has experienced rapid growth, facing numerous challenges due to the heterogeneity of devices and the great diversity in network protocols and operating systems. This leads to a gap between the knowledge required for the installation of smart environments and the general understanding.

Therefore, this proposal focuses on the generation of a ubiquitous system that reduces this gap, under the idea that with hardware that, although specialised, is designed so that the incorporation of elements such as sensors or actuators is as practical as connecting a cable (in a metaphorical sense).

But it is not all about the hardware, as it is also necessary to have a program that manages device recognition at both the modular and node level, so the main idea of this article is to describe the software design of the operating system as well as its desirable behaviour.

That is why in the subsequent sections there is first a literature review on the desirable characteristics of an operating system for embedded systems and the systems that already exist, the network protocols that are used for IoT and the types of devices that make up an IoT environment, to close with the degree of intelligence in the devices.

The next section focuses on the description of the project, how it is constituted and designed at hardware, software and behavioural level, and then continues to the section of analysis and review of the operating system. It is worth mentioning that the development of the operating system is under development, for this reason there are no results, however, this section shows the virtues and disadvantages that this proposal has.

Literature review

Desirable features in an OS

In (P. Gaur et al, 2015) the desirable characteristics for an operating *system*(OS) for IoT devices are examined, because they present challenges such as limited power supply and memory resources, which is why there is a need for an efficient, flexible, portable and lightweight system that adapts to the IoT. Desirable characteristics for an OS include:

- Architecture: a monolithic, layered or microkernel type architecture
- Programming model: A model where programmers can efficiently use the system and develop easily.
- Process scheduling: Real-time, energy efficient and multitasking.
- Network architecture: Low power consumption, platform flexibility, low weight, internet enabled and IPv6 support.
- Memory management: Will depend on the type of application and lower platform support.
- Portability: It should be easy to port to different hardware platforms.

In turn, the author (P. Gaur et al, 2015) reviews the existing operating systems and among them RIOT, FreeRTOS and μClinux stand out for sharing similar characteristics such as: being multithreaded, programmed in C/C++ and having full TCP/IP support.

In (Borgohain et al, 2015) mentions that the challenges that OS for IoT devices generally present is the limitation of resources, this means that an OS for IoT requires few kB of RAM, as well as operating with low power consumption. These OSs come built with a number of pre-installed and pre-integrated applications, drivers and network protocols.

Network protocols for IoT

There are a large number of protocols designed or used for IoT applications, such as Wi-Fi, Bluetooth, mobile network (3G, 4G, 5G), ZigBee, Z-wave, STOMP, XMPP, MQTT, CoAP, AMQP, Websockets protocols (Nuratch, S., 2018). Choosing the appropriate protocol can be difficult, but one of the key challenges is to make machine-to-machine communication in constrained networks efficient (Heđi, I. et al, 2017).

Most IoT OSs offer the full IP stack to manage the network, offering standards such as UDP (*User Datagram Protocol*), TCP (*Transmission Control Protocol*) and even HTTP (*Hyper-Text Transfer Protocol*) (Zikria, Y. B. et al, 2019).

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

IoT devices

It is (Garcia, C.G. et al, 2017) who introduces the concept of *Smart Object* to devices that are able to connect to the Internet, collect data, function as a sensor or actuator and have a certain degree of intelligence. It is different from the concept used in (Zhang, Y. et al, 2004) who refers to a *smart sensor* as a device that combines elements of sensing, information processing and communication technology, where this concept leaves out actuators.

In addition, a *Smart Object* can be seen as part of a WSN (*Wireless Sensor Network*) as a *wireless sensor node*, as it is composed of a microcontroller, a transducer, timer, memory and ADC (*Analog-Digital Converter*) according to (Farooq, M. O. et al, 2011).

IoT devices are also described as *IoT* devices that despite their limitations in power and memory (Jaskani, F. et al, 2019) include a physical layer, an interface and an IP (*Internet Protocol*) address as suggested by (Javed, F. et al, 2018).

Thus, it is possible to indicate that there is no universal name for these devices, there is also no homogeneity in hardware, which can be divided into two categories: high-end and low-end. The high-end category includes devices such as SBCs (*Single Board Computers*) and *smartphones* and the low-end category includes devices that are more limited in terms of resources (Hahm, O. et al, 2015).

It is (Roy, S. K. et al, 2019) who propose a *Plug-n-Play* (PnP) solution capable of integrating third-party embedded sensors with *IoT devices* without prior information from the sensors, this achieved through identifiers that facilitated hardware recognition.

Degree of intelligence

Within smart environments, devices vary in their degree of intelligence, while for (G. G. Meyer et al, 2009) intelligence is distributed in three dimensions, where (Garcia, C.G. et al, 2017) helps to expand the concept. These three dimensions are as follows:

1. Level of intelligence: given by the ability to handle (fundamental) information, notify problems and decision making.
2. Location of intelligence: is to discern where the intelligence is located, either in the network (outside the device), in the object itself, or combined.

Level of aggregation of intelligence: which refers to the level of divisibility of the components.

With this classification, it is possible to determine how intelligent a device or system is, because depending on the characteristics it fulfils, the graph in Figure 1 can be used to help visualise the position of intelligence in these three dimensions.

Box 1

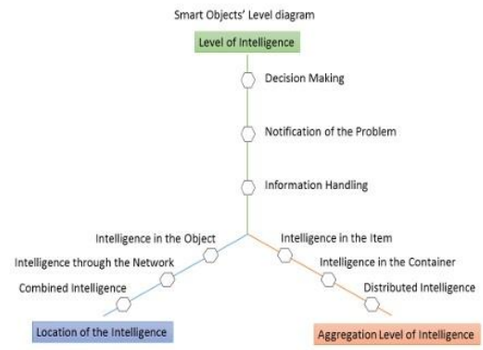


Figure 1
Dimensions of intelligence
Garcia, C.G. et al, 2017

Project description

General

It is (Tanenbaum, A. & Bos, H., 2015) who define an Operating System as the fundamental software that manages the hardware and software resources of a computer system, facilitating the execution of applications such as memory management, process control and management of input and output devices.

This is why this operating system, which from this point on will be referred to as AIOS (***A**mbient **I**ntelligence **O**perating **S**ystem*), is built from specialised hardware. In other words, AIOS is built from a *hardware* concept that enables its operation, because it will be complex to implement AIOS on existing *hardware*. The AIOS *hardware* uses the ESP32 SoC (*System on a Chip*) for the various features it offers, such as wireless connectivity via Wi-Fi and Bluetooth already integrated, 520 kiB of RAM and 448 kiB of ROM, moving it away from being a device limited in memory and its low acquisition cost.

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

It is with this SoC that we seek to move away from the idea that an *IoT device* is limited in resources and implement this OS, which meets the key features of such a device: composed of a microcontroller, a transducer, a physical interface and IP protocol for internet connectivity, which collects data and serves as a sensor or actuator.

On the other hand, the AIOS *software* design idea aims to meet two key requirements: easy integration of new devices and automatic recognition of the type of hardware that has been integrated. The AIOS *software* should be considered as such the OS of the system, which in addition to seeking to meet the aforementioned requirements, should perform the basic functions of an OS which are resource management, process control and *hardware* management. Considering that being also an IoT OS, it must satisfy the characteristics of such an OS which can be summarised as being prepared for wireless connectivity despite the limitations in resources and energy.

Hardware description

In relation to the generalities, the main *hardware* is based on the ESP32 SoC, which in addition to the aforementioned features, is integrated to handle protocols such as SPI, I2C, UART, PWM, ADC and others.

The proposed *hardware* is categorised in two forms, by circuitry:

- *Motherboard*: It consists of a PCB board (figure 2) that includes the ESP32 SoC, and physical connections to attach to the *module.*

Box 2

Figure 2
Motherboard concept
Source: *Own elaboration*

Module board: Another type of PCB consists of an embedded sensor or actuator and subject to the type of peripherals that allow it to be attached to the *motherboard*. There are three types of this type:

- *Gateway module*: composed of hardware useful for *Gateway Device* (see *by functionality*).
- *Sensor module*: composed of a sensor type transducer.
- *Actuator module*: consisting of an actuator type transducer.
- *Power board*: Refers to the PCB that will be used to power the motherboard and module board assembly.

By functionality:

- *Gateway Device*: This device consists of a *motherboard* and a *gateway module*. This device is the first to be configured, managing the main communication and configuration of the system.
- *End Device*: This device consists of a *motherboard* and a *sensor module* or an *actuator module*.
- *Auxiliary Device*: This is a sub-type of *Gateway*, but without a *gateway module* attached.

Due to the modular nature of the design, it allows the system to receive upgrades without the need for major modifications to the system.

Box 3

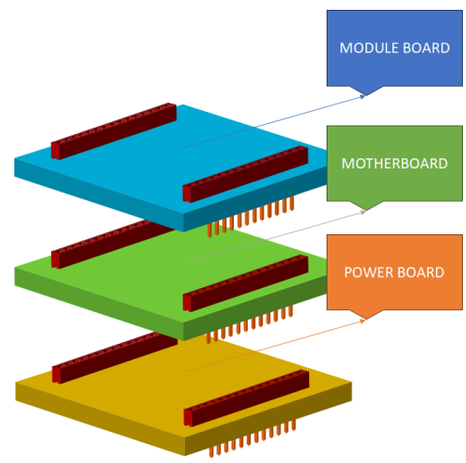


Figure 3
Hardware stacking concept
Source: *Own elaboration*

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

The concept suggests that the three types of *hardware* per circuitry can be connected as a *stack* (Figure 3).

Software description

The *software* was designed on the basis of a layered model that takes inspiration from the OSI model (***O**pen **S**ystems **I**nterconnection*) to name some of its layers, however, the functionality and purpose is completely different, Table 1 shows the relation of the layer name to the OSI model.

Box 4

Table 1

Comparison of names between the OSI model and capas of the AIOS library.

AIOS Layer	OSI Model	AIOS Main Header
Application	*Application*	AIOS_app
Presentation	*Presentation*	AIOS_os
Session	*Session*	AIOS_mid
Link	*Link*	AIOS_mod
		AIOS_com
Physical	*Pysical*	AIOS_hard
		AIOS_net
Global	*N/A*	AIOS_glob

Source: *Own elaboration*

The reason goes from the understanding that each layer manages a section of the process and the structural organisation of the library.

In this way, the structure of the library ensures that each section is dedicated to a proportional part of the necessary tasks to be executed. Figure 4 shows the sections named in Table 1, together with the tree structure of the library's headers.

In the following, we will briefly describe what each section consists of and what functions it performs, starting from the bottom to the top of the hierarchy, as this will allow a better understanding of each section.

In *Global*, there are the global resources of the program, this goes from variables, structures, classes and global objects, as well as including macro definitions and specific type elements.

In the *Physical* layer, there are two *headers*, *Hardware and Network*, both of which, despite being in the same layer, are different in functionalities and processes.

Hardware manages *hardware* protocols such as SPI, I2C, ADC, DAC, CAN, PWM and UART.

The *Network* layer manages Wi-Fi and ESP-NOW protocols, i.e. network protocols.

The *Link* layer is also made up of two *headers*, *Modules* and *Communication*, where the former is in charge of the *Module Board* type *hardware*, as the type of *hardware* protocol to be used is specified in each module. *Communication* includes protocols that are mounted on a TCP/IP network, which explains why there is a hierarchy above *Network*, these protocols are: UDP, MQTT and HTTP.

The *Session* layer includes the *Middleware*, which is the part of the *software* that links the two generated branches, as well as file management using an SD module or Flash memory.

The *Presentation* layer includes the *Operating System*, which handles general operating system functions and mechanics.

And finally the *Application* layer is the one that finishes abstracting the contents of the operating system for the user to just apply and execute.

It is with this structure that the OS was programmed and manages to satisfy the required functionalities.

Box 5

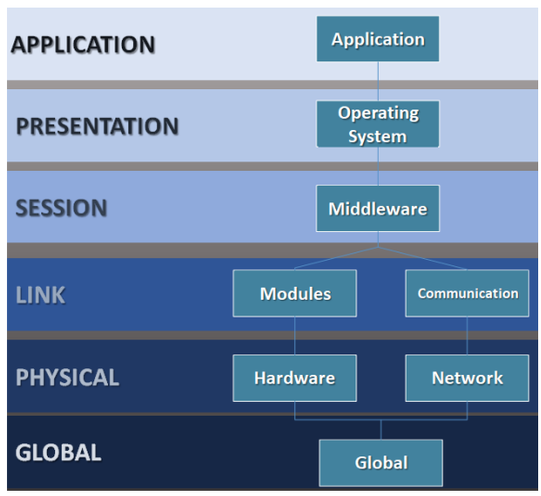


Figure 4

Structure of the AIOS library

Source: *Own elaboration*

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

Behavioural description

After knowing the basic elements of the system such as the types of *hardware*, and *hardware*, network and communication protocols that the system contemplates, it is possible to describe in depth how the system works in general terms.

The device follows a general sequence to achieve rudimentary operation, so the first step is the installation of the *hardware*.

In this first step, human intervention is necessary, where the user takes a *motherboard*, a *gateway module board* and a *power board*, couples them together and so has built a *Gateway Device*, at power up, this device will execute the first start-up actions.

The *Gateway Device* will load configurations and examine the SD card (which has documents that are necessary for start-up) and if everything is in order, it will show on the screen that the user must connect to the Wi-Fi network (*Access Point* mode) that the device generates, a network that does not connect to the internet, but to a web portal that allows entering home network configurations, as if it were an internet *router*.

When the user connects to the *Access Point* network, whose credentials (SSID and password) are displayed on the device's LCD screen, the user must then open a browser and enter the IP address that is also displayed on the LCD screen.

In the web portal, the user can enter the credentials of his home network, there is a button that shows or hides the expert mode settings (fixed IP address, internet gateway address, subnet mask, among others) which may or may not be entered by the user. When entering the data, the device will save the data and reboot to connect to the entered network, if the user only enters the SSID and password, the device will get the other data automatically.

When the *Gateway Device* is ready, it proceeds to wait for new devices in the area, this is where the user must initialize an *End Device*, assuming the user has a temperature sensor, then it would attach a *motherboard*, the *module board* (which would be the temperature sensor) and a *power board*.

The *End Device* will generate a *broadcasting* message looking for the *Gateway Device*, when it finds it, both will start a *handshaking* protocol to achieve communication between both devices.

The user must repeat the same with each *End Device* he wants.

The last part is to manage how you want these devices to be activated, this requires an MQTT server to be configured to which the *Gateway Device* will be launching the information collected from the *End Deviece*.

This part of the project is not in the pipeline, but it is suggested that a PC program that is subscribed to the general topic where the *Gateway Device* is publishing, can manage all these devices. The OS will therefore also be listening for messages from the server in order to manage these instructions.

Communication protocol

There are four types of messaging events that occur in AIOS, and three types of actors in each event. The three actors are: *End Device* (for practical purposes, this will be the ‘client’), *Gateway Device* or an *Auxiliary Device* (the ‘manager’) and the *Server* (MQTT server or simply ‘server’), the latter being understood as an interface that exists between the devices and the user itself.

The first type of event (Figure 5) is the one that occurs when a client encounters a manager. The client sends a request message, which includes information relevant to its configuration (such as MAC address, hardware type, etc.) and the handler will register the device in the system itself and in the server, reply to the client with its information, then the client confirms, waits for the handler's confirmation, and from this point onwards the three actors will be linked in a continuous way.

Box 6

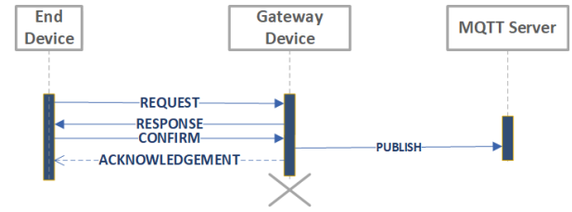


Figure 5
First contact communication
Source: *Own elaboration*

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

The second type of event can be divided into two types, as it refers to the general exchange of data from a sensor to the system, or from the system to an actuator. Thus, if a client has a sensor attached (Figure 6), it sends the sensor data to the manager and the manager publishes it on the server, and then confirms to the client that it has been sent.

Box 7

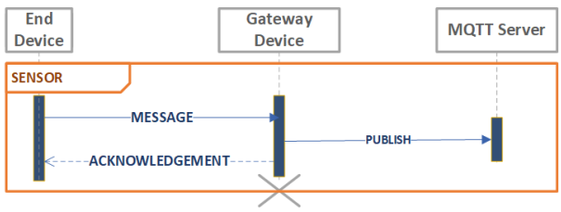


Figure 6
Communication from the sensor
Source: *Own elaboration*

Box 8

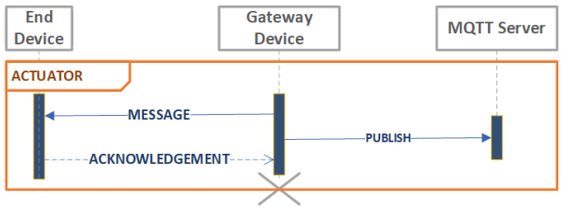


Figure 7
Communication to the actuator
Source: *Own elaboration*

On the other hand, an actuator expects to be activated by some variable (figure 7), it is then the manager who generates this message to the client with the actuator, in addition to publishing on the server, and would expect a confirmation message from the client with the actuator attached.

The third and last type of event originates from the server (figure 8), consider that the user wants to link two clients, a sensor and an actuator, wants the actuator to be activated when the sensor variable reaches a value, or the user wants to change a configuration such as the network to which the whole system should be connected. For both cases, the server originates the message and publishes it in the manager topic, it is the manager who sends the instructions to those involved (this means that the instruction may or may not be addressed to him or the clients), on receiving confirmation from those involved, it then publishes to the server that the required changes have been made.

Box 9

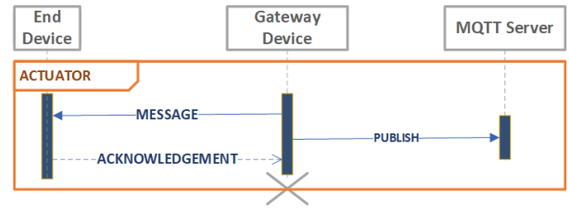


Figure 8
Communication by system instruction.
Source: *Own elaboration*

In general terms, the types of message events can be summarised as: first contact, regular operation and system instructions. All three types of events will always involve the three types of actors: client, manager and server.

Analysis and review of the proposal

AIOS versus desirable features

The AIOS operating system is based on a layered and modular kernel, where each layer is in charge of specific tasks and modular because although all the code exists, only what is required is executed. As stated by (P. Gaur et al, 2015) 'modular architecture proves to be better than monolithic architecture, as the failure of one module does not lead to the failure of the whole system'.

As for the programming model, it seeks to be one that allows efficiency and use the system, it must also allow the programmer to focus and increase their productivity, AIOS is built with C++ because it is object-oriented programming and thus be able to manage abstractions and use the hardware as efficiently as possible.

In terms of process scheduling, AIOS was developed in an event-driven system, especially for start-up tasks, however, in regular operation, it takes advantage of functions that are executed asynchronously, and others that are executed in parallel, hence, it is a combination of both models.

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

In terms of network architecture (P. Gaur et al, 2015) proposes that communication should be in a standard that allows communication over the internet without complications, light and reliable, but at the same time with IPv6 protocol. AIOS for its part ignores some of these conditions, remembering that the SoC, ESP32 is not limited in resources as IoT devices classified as low-end, then it takes advantage that AIOS is launched in protocol with full TCP/IP, IPv4 and protocols such as MQTT, HTTP and UDP, being more versatile.

The memory management is mentioned that many IoT OS do not include a memory management unit (MMU) or floating point unit, which AIOS does, plus the operating system is mounted on the flash memory of the SoC and manages files using SD card (for *Gateway Device* only) and SPIFFS (flash memory, on the other devices).

Box 10

Table 2

Comparison of AIOS vs. operating system features

Category	Ideal	AIOS
Architecture	Monolithic	Layered and modular
	Layered	
	Micokernel	
Model of programme	Assembler	C++
Process programming	Real-time	Per event and real time
	Multitasking	
	Multithreading	
Network architecture	IPv6	IPv4, internet connectivity, MQTT, HTTP and UDP
	Internet connection	
	Low weight	
Management of	Not specified	SD, SPIFFS, Flash
memory	Must be	Currently only for ESP32

Own material with information from Gaur et al., 2015

The issue of portability is not contemplated, which is a point against it, as its development is oriented towards ESP32. However, the nature of the system will allow for portability later on.

As we have already seen, AIOS offers not only a type of connectivity to link the devices, as this architecture idea intends the operating system to manage the communication more efficiently.

Gateway devices are able to generate Wi-Fi networks on their own, and then connect to the internet, and take advantage of this function to also connect to an MQTT server and manage system messages to the user.

On the other hand, communication between system devices takes advantage of the ESP-NOW protocol, which allows better management of messaging between devices by allowing bidirectional linking, broadcasting, and intensity measurement tools to be used for communication.

It leverages the UDP protocol for inter-device messaging, which, while not as reliable as TCP, is a faster UDP protocol.

In addition to the fact that AIOS is modular, the integration of other protocols into the system can be expected to be feasible.

AIOS and IoT devices

Compiling the concepts seen in the introduction section of IoT Devices, each concept is compared with respect to the features of AIOS devices.

Table 3 shows the characteristics of a Smart Object (SO), Smart Sensor (SI), Wireless Sensor Node (NS) and IoT Device (DI), abbreviated as such to be shown in the table.

And it is compared to the three AIOS device types: Gateway Device (G), Auxiliary Device (A) and End Device (E). Where a Gateway Device and an Auxiliary Device have access to internet and communication technology, collect data from sensors (indirectly) as well as process them, have a degree of intelligence, a microcontroller, physical interfaces and memory.

End devices, although they can also connect to the internet, do not actually do so for AIOS, nor do they process the information, as they are only receivers of it, and although they do involve some degree of processing, in practical terms, they only use it.

AIOS devices are also plug-n-play, thanks to hard-ware recognition by the device itself, and then they are able to communicate with each other automatically. The only missing feature is the ability to automatically indicate what you want to do with a variable and when to activate a certain element in the system, a complexity given that AIOS is designed to be implemented in a variety of environments that handle their own variables and actions.

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

Box 11

Table 3

Comparison between Smart Object (SO), Smart Sensor (SI), Wireless Sensor Node (NS) and IoT Device (DI) with AIOS devices (G: Gateway Device, A: Auxiliary Device, E: End Device); Green (yes), Yellow (indirectly), Orange (has it, but not used), Red (No), Red (not used).

Features			AIOS devices		
			G	A	E
Wireless comm.	Internet (SO, DI)				
	Technology of technology (SI)				
Data	Collection (SO)				
	Processing (SI)				
Level of intelligence (SO)					
Physical characteristics	Microcontroller (NS)				
	Transductor	(NS)			
		Sensor (SO, SI)			
		Actuator (SO)			
	Interface	Physical layer (DI)			
		ADC (NS)			
	Memory (NS)				

Source: *Own elaboration*

AIOS and the degree of intelligence

For this section, what was mentioned in the section on the degree of intelligence in the introduction is taken up again, there are three dimensions, to which one point will be assigned for each characteristic, thus, if you comply with the three individual characteristics of each dimension, your score will be three, the maximum. Table 4 shows the intelligence dimension and the characteristics, because level three for the dimensions "location of intelligence" and "aggregation of intelligence" is valid for levels 1 and 2, the three points are automatically assigned.

Analysing the AIOS system on the intelligence level, it meets the criterion of information handling, because it is constantly acquiring and processing data; the system does not report problems, both internally and externally, internally it processes circumstances such as unrecognised hardware, and without the possibility of communication, externally, it is also constantly listening to other devices, so it reports if there are inactive devices, therefore, it is awarded the point. As for decision making, this is also considered to occur because the device will perform hardware recognition and when looking for which device to connect to, it counts as decision making, since it is not the user who tells it these things. So on the intelligence level it gets the three points.

Box 12

Table 4

Dimensions of intelligence levels

Dimension	Level 1	Level 2	Level 3
Level of intelligence	Handling information	Notification of problems	Decision-making
Location of intelligence	On the web	The object itself	In the network and object
Aggregation of intelligence	In the embebido	At the container	Distributed

Source: *Own elaboration*

As for the dimension of the location of intelligence, it is considered to be in the network and in the object, because it is the object that recognises elements such as hardware and it is through the network that it communicates information and data. It is the device that makes decisions such as which hardware to read or activate, and it is thanks to the use of the network that it can capture other data that makes it function properly. Therefore, it is awarded all three points.

Finally, the dimension of intelligence aggregation becomes a bit more complicated to analyse, as there is an ambiguity, since the category "in the embedded" indicates that the device is unique and all the necessary elements to achieve intelligence are on the same board, while "as a container" indicates that it requires additional hardware to make this happen. In AIOS, it is concluded that it complies with both, since it is a modular hardware design, indicating that it requires additional hardware to execute the necessary actions. It is also true that the Auxiliary Device devices work without additional hardware, since the ESP32 is a SoC that integrates several of its own characteristics that are used by AIOS to make it work. With the above it can be concluded that AIOS is a container system, which in turn uses a SoC which is an embedded system, making AIOS as a system with distributed intelligence aggregation.

In Figure 8 a radar type graph is used that shows the three dimensions of intelligence and each point gained would be the level achieved, in the case of AIOS it achieves the three points in each dimension.

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

Box 13

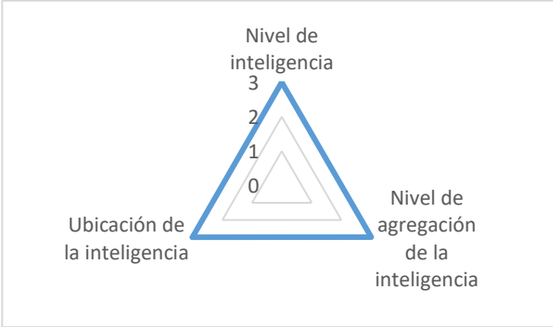


Figure 8
AIOS degree of intelligence
Source: *Own elaboration*

Acknowledgement

We thank CONAHCyT and TECNM, Campus León, for their support in the development of this research.

Conclusions

To conclude, AIOS is built as an intelligent operating system, taking advantage of functionalities that seek to help in the generation of intelligent environments, looking even as an IoT application, or a wireless sensor network or cyber-physical system, AIOS tries to offer a system where the technical knowledge required is the minimum to operate properly, being this an advance in the search for the ubiquity of systems, being a pervasive system which provides humans the power of computation for their needs and comfort.

Although there may be more variants to achieve the same goal, the development of AIOS made primary features such as modularity to provide for expansion and growth of the system, and it can be used in any type of intelligent environment that may be required, such as hospitals, factories, home use, etc.

It is clear that it is an immense system that, although still under development, the culmination of this will allow the application of more techniques and technology such as the generation of natural interfaces, application in artificial intelligence or even for data analysis and Big Data. Finally, the ambition of the project and the feasibility of its development are highlighted, where the completion of the code, the complete development of the *hardware* and the interfaces that bring the user closer to the practical use of this operating system are envisaged.

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Declarations

Conflict of interest

The authors declare no interest conflict. They have no known competing financial interests or personal relationships that could have appeared to influence the article reported in this article.

Author contribution

Moreno, Paul: Methodology, software, hardware design, implementation, investigation, validation, project management.

Baltazar, Rosario: Project idea, supervision, paper review, validation, project administration, methodology, conceptualisation, research method.

Casillas, Miguel Ángel: Design validation, software validation, software review, supervision, formal analysis, resources acquisition.

Availability of data and materials

Data sharing is not applicable to this article as no new data were created or analysed in this study.

Funding

This work has been funded by CONAHCyT [$348,528.42 MXN]

Acknowledgements

We thank CONAHCyT and TECNM/ITLeón for the support they provided throughout this research.

Abbreviations

ADC	Analog-Digital Converter
AIOS	Ambient Intelligence Operating System
AMQP	Advanced Message Queuing Protocol
AP	Access Point
CAN	Controller Area Network
CoAP	Constrained Application Protocol
DAC	Digital-Analog Converter
HTTP	Hyper-Text Transfer Protocol
I2C	Inter-Integrated Circuit
IoT	Internet of Things
IP	Internet Protocol
IPv6	Internet Protocol version 6
kB	kilo Byte
LCD	Liquid-Crystal Display
MAC	Medium Access Control

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11

MMU	Memory Management Unit
MQTT	Message Queue Telemetry Transport
OS	Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
PCB	Printed Circuit Board
PnP	Plug-n-Play
PWM	Pulse-Width Modulation
RAM	Random Access Memory
ROM	Read-Only Memory
RTOS	Real Time Operating System
SBC	Single Board Computer
SD	Secure Digital
SoC	System on Chip
SPI	Serial Peripheral Interface
SPIFFS	SPI Flash File System
SSID	Service Set Identifier
STOMP	Streaming Text Oriented Messaging Protocol
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
Wi-Fi	Wireless Fidelity
WSN	Wireless Sensor Network
XMPP	Extensible Messaging and Presence Protocol

References

Antecedents

P. Gaur y M.P. Tahiliani, "Operating Systems for IoT Devices: A Critical Survey", 2015 IEEE Region 10 Symposium, Ahmedabad, India, 2015.

Borgohain, T., Kumar, U., and Sanyal, S. (2015). Survey of operating systems for the iot environment. arXiv preprint arXiv:1504.02517.

Zikria, Y. B., Kim, S. W., Hahm, O., Afzal, M. K., and Aalsalem, M. Y. (2019). Internet of Things (IoT) operating systems management: Opportunities, challenges, and solution. Sensors, 19(8).

Zhang, Y., Gu, Y., Vlatkovic, V., and Wang, X. (2004, June). Progress of smart sensor and smart sensor networks. In Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No. 04EX788) (Vol. 4). IEEE.

Javed, F., Afzal, M. K., Sharif, M., and Kim, B. S. (2018). Internet of Things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review. IEEE Communications Surveys and Tutorials, 20(3).

G. G. Meyer, K. Främling, and J. Holmström, "Intelligent Products: A survey," Comput. Ind., vol. 60, no. 3, Apr. 2009.

Basics

Nuratch, S. (2018, July). Applying the MQTT protocol on embedded system for smart sensors/actuators and IoT applications. In 2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (Ecti-con). IEEE.

Heđi, I., Špeh, I., and Šarabok, A. (2017, May). IoT network protocols comparison for the purpose of IoT constrained networks. In 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE.

García, C. G., Meana-Llorián, D., and Lovelle, J. M. C. (2017). A review about Smart Objects, Sensors, and Actuators. International Journal of Interactive Multimedia and Artificial Intelligence, 4(3).

Farooq, M. O., and Kunz, T. (2011). Operating systems for wireless sensor networks: A survey. Sensors, 11(6).

Hahm, O., Baccelli, E., Petersen, H., and Tsiftes, N. (2015). Operating systems for low-end devices in the internet of things: a survey. IEEE Internet of Things Journal, 3(5).

Andrew S. Tanenbaum and Herbert Bos. Modern Operating Systems. Pearson, 4 edition, 2015.

Supports.

Roy, S. K., Misra, S., and Raghuwanshi, N. S. (2019). SensPnP: Seamless integration of heterogeneous sensors with IoT devices. IEEE Transactions on Consumer Electronics, 65(2).

ISSN: 2531-2197
RENIECYT-CONAHCYT: 1702902

Moreno, Paul, Baltazar, Rosario and Casillas, Miguel Ángel. [2024]. A proposal for an IoT operating system for plug-n-play wireless sensors. Journal Computer Technology. 8[20]-1-11: e40820111.
DOI: https://doi.org/10.35429/JCT.2024.8.20.4.11