IOT OPERATIONS AND INTEROPERABILITY

Abstract

Internet of things is a global network architecture that was created as a result of the interconnection and communication of numerous smart devices in the physical world over existing internet infrastructure in recent years. According to studies, over the previous 6-7 years, solutions for a variety of devices and Internet of Things platforms have made significant advancements. Interoperability problems do, occur because solution offers a distinct Internet of Things architecture, devices, APIs, and data formats. These interoperability problems are largely caused by the difficulties of connecting non-interoperable Internet of Things devices to various Internet of Things platforms, vendor lock-in, and the inability to create Internet Of Things apps that deliver cross-platform and/or cross-domain data. The widespread deployment of Internet of Things technology is hampered by all of these issues. The ability of numerous IOT systems from various suppliers to cooperate has been improved through a number of initiatives by academics, corporations, and standards groups. Traditional operating systems now used on Internet hosts, as well as conventional operating systems, are unable for simultaneously meet the specifications of such a diverse array of devices. Therefore, a new, consistent operating system is needed to eliminate duplication in the price of creating and maintaining IOT products. In this article, the prerequisites for an IOT operating system are examined.

Key Words: IOT applications; IOT requirements and operating systems; improve IOT interoperability; interoperability in IOT Taxynomy.

Authors

Babitha Gaikwad G
Electrical And Electronic Engineering
Presidency University
Babithagaikwad372@gmail.com

Bibang Gwar Basumatary
Research Scholar
Department of Civil Engineering
Presidency University
Bengaluru, India
bibang1999@gmail.com

Nakul Ramanna
Professor and HOD
Department of Civil Engineering
Presidency University
Bengaluru, India.
nakul@presidencyuniversity.in

I. INTRODUCTION

The Internet Of Things, which Kevin Ashton utilized for the first time in 1999, has recently emerged as a cutting-edge technology in number of industries. The Internet of Things is the network of physical locations and things that are connected via the Internet. According to this scenario, a technological revolution will connect digital and analog objects to both the current Internet infrastructure and to one another. The Internet of Things European Research Cluster (IERC). The Internet of Things is an adaptable, self-configuring worldwide network architecture due to standardized and cooperative communication techniques. With the Internet of Things, both real-world and digital items have identities, physical traits, and virtual personalities, as well as intelligent interfaces, enabling them easy incorporation into the information network. Academics and business executives have emphasized the significance of the IOT interoperability dilemma. Standardization is being used by the sector that will deal with IOT interoperability issues. Numerous initiatives have been started to ensure Internet Of Things systems, networks, and services as well as data formats produced by other manufacturers. We conducted research on the past, present, and anticipated advancements of enabling technologies and solutions in order to better inform readers about the current situation and anticipated developments in IOT interoperability.

This study has two advantages. First, let's have a look at the general requirements for IOT device software. It demonstrates that no existing operating system can meet the numerous requirements of IOT systems, including limitations, a wide variety of network stacks, autonomy, and real-time constraints. There have been numerous attempts to adapt present operating systems for the IOT, but it has been impossible to include crucial components like maximum energy efficiency or robust real-time assurances since they have an impact on every aspect of the system.

II. IOT REQUIREMENTS

1. **Use Cases:** Several projects and a substantial scientific community are now addressing the IOT topic. It conducts research and development on a CPS that uses sensors to improve security and safety in populated regions and close to important infrastructure. To precisely target area monitoring in airports, SAFEST combines two different types of technologies.

   - A system of visual and audio surveillance that keeps an eye on large crowds to provide warning in the event of unforeseen circumstances.
   - A perimeter security system that searches for unalgorithms industries using distributed event detection algorithms.

   At this point, a problem occurred because, as will be detailed further below, present operating systems are either unable to make making use of the robust board's skills or on the confined board, cannot run. This necessitates the simultaneous use of multiple operating systems, requiring the creation and upkeep of duplicate application code that is shared over a network of such devices.

2. **Software Characteristics:** The project states that typical situations involve a variety of heterogeneous devices with varied levels of complexity. This implies that the software that runs on such devices must adhere to strict guidelines. Restricted hardware is the first category of demand. The requirement for these systems to function independently is covered by a second category. A third category of requirement assesses the system's usefulness from the perspective of a developer.

III. OPERATING SYSTEM

As we saw with the network stack in the previous section, an operating system needs to be carefully built in every way if it is to respond to the limitations of ordinary IOT devices. In this section, we will examine how operating systems should be designed for the context of the IOT, also to analyze and contrast current operating systems.

1. **Characteristics:** Different operating system types can be distinguished by a wide range of characteristics. The kernel's structure is one of the most fundamental design elements. There are three different ways to build the operating system: monolithically, layeredly, or with the microkernel architecture. This decision has a substantial impact on the overall structure of the system. Although building an operating system from the ground up using a monolithic kernel is the most fundamental approach is lacking typically resulted in a complicated framework that is flexible harder to grasp since the system grows. The system's hierarchical division is facilitated by the layered model. The developer must choose how closely user space and the kernel are separated. A microkernel takes modularity a step further by partitioning the whole operating system into manageable, well defined modules, with just a tiny subset of operations taking place in kernel mode. This technique increases system stability since it prevents system crashes due to errors in specific components. Another crucial design factor is the scheduler. The system's capacity to support real-time attributes, numerous priorities, and various user participation levels is inextricably linked to the scheduling method that is implemented. The third and most crucial design element is the model for programming. a number of operating systems run every procedure in the same context when memory address space segmentation is not present. Multi-threading is a feature of several other operating systems that allows each process to execute in its own thread with its own memory stack. The programming model and the programming language are interconnected. There may be a major impact on both the programming languages used to create operating systems and the programming languages accessible to application developers.

2. **Comparision of Operating System:** Tiny OS and Contiki are the two WSAN device operating systems that are most frequently utilized. Both employ different file systems, shells, device drivers, algorithms, protocols, and tools. The majority we now analyze contemporary operating systems that are purportedly suitable for running IOT devices while keeping in mind the essential design attributes and needs listed in Section II. To do this, we chose representative operating systems from both widely-used full-featured operating systems found on Internet hosts and embedded WSAN operating systems.

   For more conventional Internet-connected devices, Windows, various UNIX variants, and Linux are the most popular operating systems. Linux will be used as example full-featured given that it is open source and offers assistance a variety equipment platforms. The majority of the claims made here hold true for both the UNIX and Windows versions.

It investigate these operating systems' characteristics in connection to the previously described design categories. Monolithic kernels are used by both Linux and Tiny OS, however Contiki was built in a way that more closely resembles a layered architecture. To provide a single static binary, Tiny OS combines many essential parts. The components interact with one another via events and commands, and they expose one or more interfaces. Device drivers can be set up as modules even if the Linux kernel is a monolith.

With the help of this method, an application's requirements can be exactly met by configuring a Linux system. A faulty driver can still cause the system to crash even though these modules can be loaded and unloaded while the system is in operation. Operating system features like connection, device drivers, and sensor data management are provided by Contiki. The device driver loader, the proto threading system, and the u IP stack are additional components of the Contiki core.

Purely event-driven scheduling is used by Contiki, which is similar to Tiny OS's FIFO scheduling strategy. Their scheduling plans are made to handle straightforward situations, such asynchronous sensor disruptions. Linux now uses the red-black tree-based Completely Fair Scheduler (CFS), which makes sure that processing time is distributed fairly. The objectives of this scheduler are to increase CPU usage and interaction performance.

Even though all tasks are completed inside the same context in the event-driven programming models used by Contiki and Tiny OS, they do offer a very small amount of multi-threading. TOS Threads, which employ a cooperative threading paradigm and demand that a program actively cede the CPU, are introduced by Tiny OS. Contiki provides proto threads as an easy-to-use, stack-free multi-threading technology. Process synchronization amongst proto threads is not possible since events run to completion.

Contiki employs a simplified version of the C programming language that lacks some jargon. C, a dialect of C, is used to create Tiny OS. On the other hand, Linux was created in C and supports a broad variety of scripting and programming languages.

IV. INTEROPERABILITY IN IOT: A TAXYNOMY

The Contiki and Tiny OS programming models are event-driven, meaning all tasks are carried out inside the same context, however they do support a very limited amount of multi-threading. Tiny OS, which make use of a joint threading a program must explicitly cede the CPU in a system. Contiki offers proto threads straightforward, stack-free implementation of basic multi-threading. Process synchronization amongst proto threads is not possible since events run to completion. To the fullest extent possible, the numerous IOT components should effectively communicate and work with one another. As shown in Figure 1, there are several ways to look at IOT interoperability, including from the perspectives of devices, networks, syntactic interoperability, semantic interoperability, and platform compatibility.

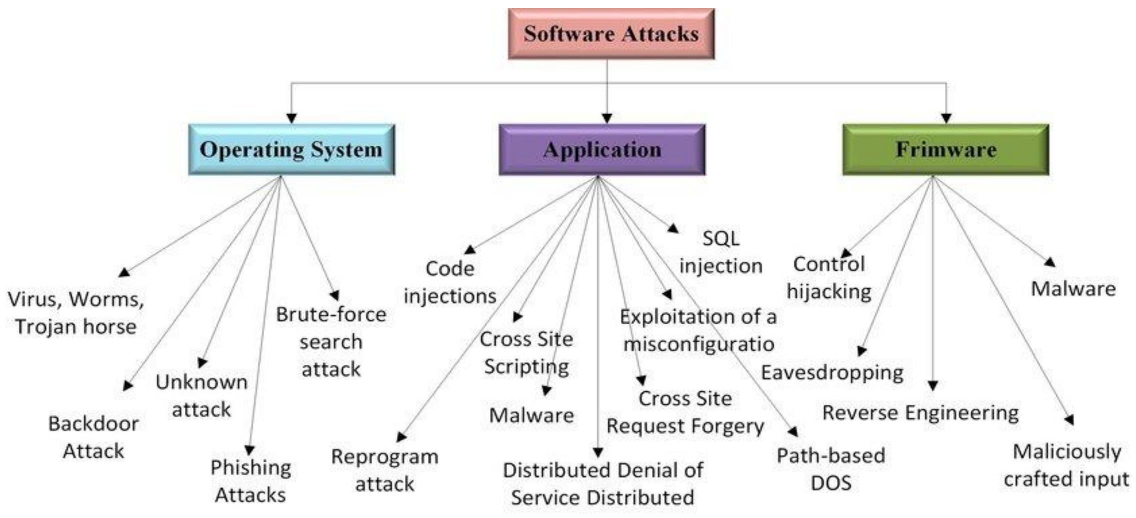

Figure 1: IOT Taxnomy

1. **Device Interoperability:** The Internet of Things has a far larger number of devices than the traditional Internet. These gadgets, often known as B smart items or things, can be pricey or reasonably priced. Processing power is abundant in smartphones and other high-end IOT devices like the Raspberry Pi. Low-end IOT devices, in contrast, have less energy, computational hosts with more energy and communication skills. Examples include inexpensive actuators and sensors, Arduino, Open Mote. The microcontroller (MCU) between manufacturers and models, there are significant differences in architecture and other important system components like as processor speed, RAM, communication protocols, and battery capacity. IOT needs vary, which has caused a number of communication protocols to arise. The microcontroller (MCU) architecture of IOT devices, as well as key system features like processing speed, RAM, communication protocols, and battery life, vary greatly between manufacturers and models. The various requirements of IOT industries have led to the emergence of numerous communication protocols. The ability of disparate IOT devices to connect and communicate with one another a range of device interoperability refers to communication standards and protocols. The ability to integrate new devices into any IOT platform is known as device interoperability. It raises questions about:

   • Data transfer between disparate devices and communication protocols.

2. **Network Interoperability:** IOT devices will continue to operate on diverse, multi-service, multi-vendor, and heterogeneous networks. IOT devices depend on irregularity and instability, unlike desktop PCs. The architecture of microcontrollers (MCUs) and essential device system characteristics include processing speed, RAM, and protocols for communication, as well as battery power, varies significantly between manufacturers and models. Numerous communication protocols have emerged as a result of the diverse requirements of IOT sectors. Device interoperability is the ability of heterogeneous IOT devices to integrate and communicate with a variety of communication protocols and standards. The capacity to integrate new devices into an platform is known as interoperability. Information exchange between heterogeneous communication protocols and heterogeneous hardware are among its main concerns.

3. **Syntactical Interoperability:** In order for two separate IOT system components to communicate with one another, there must be syntactic interoperability, which is the compatibility of the employed format and data structure. Every resource needs a user interface that shows a structure that follows a particular schema. REST APIs and WSDL are two examples. A message must be serialized before it can be sent across a channel, and the serialization format (such as XML or JSON) must be supplied. Data in messages is encrypted by the message sender utilizing syntactic rules found in several grammars. Using the specified syntactic rules in the same or a different the message recipient decodes the delivered message using grammar.

4. **Semantic Interoperability:** In order for two separate IOT system components to communicate with one another, there must be syntactic interoperability, which is the compatibility of the employed format and data structure. Every resource needs a user interface that shows a structure that follows a particular schema. REST APIs and WSDL are two examples. A message must be serialized before it can be sent across a channel, and the serialization format (such as XML or JSON) must be supplied. Data in messages is encrypted by the message sender utilizing syntactic rules found in several grammars. The recipient of the message decodes it using the provided similar syntactic rules or alternative grammar. Different sources frequently utilize different data models and schemas that are not always compatible necessarily interoperable, even though the environment's entities may produce data in a specific format. Additionally, the info may use other measurement units and be conveyed with additional information.

V. INTEROPERABILTY DOMAIN KNOWLEDGE

Developing templates for SWOT applications without interoperable data and a set of rules is a difficult task. To accomplish so, we must be familiar with cross-domains, datasets, and IOT data rules.

VI. DESIGNING, DEVELOPING

The design phase assists developers in selecting the application that will be deployed by template using semantic web query. The developer fills the program with data and rules and arranges data to implement the application during the development stage. End users enter run time into the running level application.

VII. CLOUD COMPUTING AND IOT

Cloud computing refers to the consumption of required services from providers as scheduled ahead of time over the internet. Companies can begin with few resources and then scale up as needed. Cloud computing is a novel technology for hosting and providing services (SaaS, PaaS, IaaS, and so on), but the concept is not new; in 1960, John McCarthy predicted that computing services will be publicly available in the future.

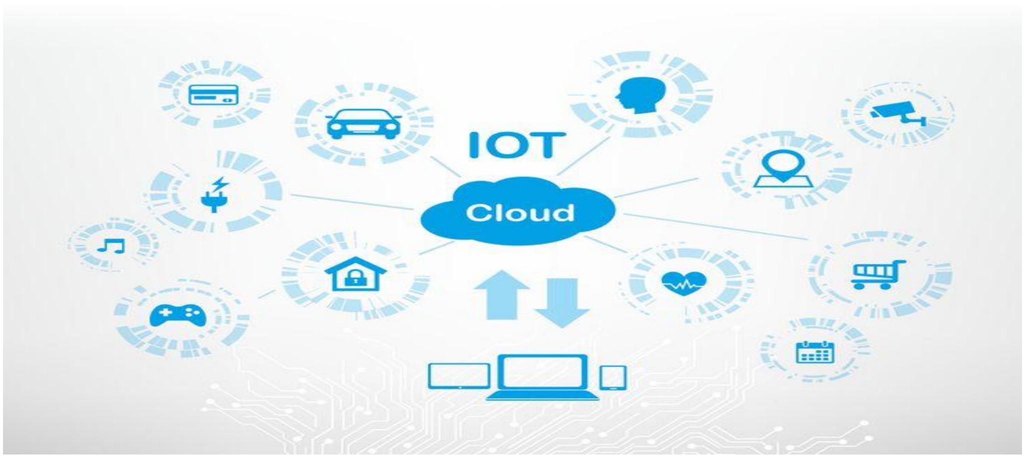

Figure 2: Cloud computing IOT

By 2020, there are expected to be 24 billion linked gadgets, surpassing the number of people. Managing that amount of data is a challenging task. Devices have limited default storage, which makes it necessary to use a permanent, dependable, and secure leasing solution that can increase data storage and processing speed. Integration of cloud computing and IOT can help resolve a number of problems, including those related to resource allocation, data integrity, security, and storage capacity.

VIII. REAL TIME OPERATING

Real Time Operating is made to both fulfill the demands outlined in Section II and fill the space we discovered between WSAN both operating systems conventional featured existing operating system on Internet hosts.

System is built utilizing a modular design to consume the least amount of memory. As a result, certain needs can be met by changing the system's configuration. A minimal amount of dependencies exist between components.

Because it doesn't require periodic events, RIOT's scheduler can be thought of as a tick-less unlike many other operating systems, scheduler. This method will ensure that the device uses as little energy as possible while maximizing the length of time in sleep mode. The computer only exits its idle state in reaction to interruptions from the outside world or the kernel. Because it doesn't require periodic events, RIOT's scheduler can be thought of as a tick-less in contrast to many other operating systems, scheduler. If no tasks are open, RIOT will switch to the idle thread. The idle thread's sole purpose is to calculate the deepest sleep level based on the utilised peripherals. This will ensure that the device uses as little energy as possible while maximizing the length of time in sleep mode. The computer only exits its idle state in reaction to interruptions from the outside world or the kernel. RIOT's scheduler can be viewed as superior to many other operating systems of as a tick-less scheduler because it doesn't require periodic events.

A Memory Management Unit are not necessary for RIOT. However, because CPU-dependent code and the kernel implementation are tightly related, RIOT can take advantage of any extra features the microcontroller provides, such as the various ARM processors offer the Vectored Interrupt Controller (VIC). The implementation, in general, abstracts away from the hardware, allowing the creation of system libraries, kernel functions, and programs that are independent of specific platforms. This is accomplished by exposing unambiguous interfaces and preserving a distinct division between hardware-dependent and hardware-independent code. The separation also makes it possible to create hardware-specific functionality without having to modify the kernel or system libraries. Function pointers and other indirection techniques that increase overhead are not used.

IX. LIMITATIONS

By reducing the distances between them, "IOT" technology aims to make physical objects more interactive so that they can exchange critical data more quickly. However, the IOT paradigm has a number of drawbacks that need apps finishing their tasks on time. One of the numerous issues with IOT is data management because all connected things generate a large amount of data every second that is challenging for the existing infrastructure to handle. Data mining needs the use of data mining technologies to process and analyze it as data volumes grow. IOT data also includes streaming data, such as location, temperature, chemical changes, and so on, in addition to normal data (plan text, tables), which is a challenge for data mining algorithms to rapidly resolve. The security risks increase with the scale of the Internet of Things network lack of encryption, unsafe online interfaces, poor software protection, and lack of software authorisation are all contributing factors, IOT presents security challenges. Developers should incorporate security tools like firewalls into their products and instruct customers on how to use built-in device security capabilities in order to address IOT security problems. IOT architecture poses a cost and performance challenge for service-based devices. A network with many linked devices has scalability problems in data management and transfer, among other areas.

X. IOT CONNECTIVITY

A variety of IOT connectivity methods can be used to connect one IOT device to another over the Internet. Both wired and wireless Internet connections are typically available. Wireless and cable communications each have advantages and disadvantages, therefore one should decide which is best for the application. We must take into account both the advantages and disadvantages of wired and wireless connecting methods before establishing an IOT system. Several factors have a role in this choice. The size of the network's IOT nodes, their location, the necessary bandwidth or data rate, the permitted maximum power consumption, and the security requirements are a few examples of these criteria. Several factors have a role in this choice. The network's maximum range, the placement of the IOT nodes, and their number, the necessary bandwidth or data rate, the permitted maximum power consumption, and the security requirements are a few examples of these criteria. IOT devices must be close to one another as well as a wired Internet access point in order for the wired IOT network to be effective. Wireless IT equipment is the standard because most IOT applications require a wired connection. IOT is made possible by wireless short-range technologies like Bluetooth, Zigbee, and even WiFi.

These technologies are particularly enticing since they benefit from the Industrial, Science, and Medicine (ISM) spectrum. To connect IOT devices to the Internet, these technologies need an IOT gateway, which is connected to the IOT physical device on one side and the Internet on the other. The IOT gateway is located at the local network's edge and has enough computing power to run small calculations there on occasion. Several Internet of Things (IOT) devices are connected to an IOT gateway in order to provide data to the Internet, as seen in Figure 3.

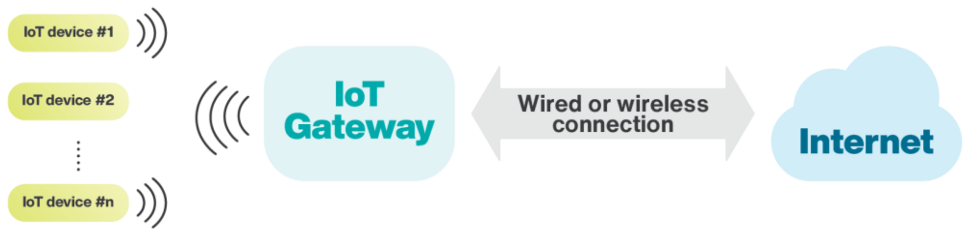

Figure 3: IOT gateway for internet connection

As a result of the rapid technological development of wireless IOT, a domain of regularly battery-powered gadgets is starting to emerge. It would be difficult to replace these batteries in many situations. Thus, one of the prerequisites for wireless IOT technologies is low power consumption. Another rationale for creating low-power IOT devices is the fact that there will be a large number of them in the future. If we can lower the power consumption of these devices, we will require less energy to power them. An IOT device must sleep if there are no tasks it needs to do in order to conserve power. Due to the fact that it consumes less energy while sleeping, the battery life of the smartphone gets longer. One of the key elements of wireless IOT is low power consumption, which explains the widespread adoption of Zigbee and Bluetooth Low Energy (BLE) in IOT applications. WiFi could be used by IOT applications that require more bandwidth but don't care about battery life.

APPLICATIONS

- Communicate and share information among them.
- For port management
- INTER-Health
- Lifestyle monitor: Medical perspective

REFERENCES

[1] Christen P, Georgakopoulos D (2014), Perera C, Zaslavsky A.
[2] Al-Fuqaha A, Guizani M, Aledhari M, Ayyash M (2015).
[3] Li S, Da Xu L, and He W. (2014).
[4] Bandyopadhyay S, Sengupta M, Maiti S, Dutta S (2011).
[5] Zeiger F (2015), Gazis V, Goertz M, Huber M, Leonardi A, Mathioudakis K, and Wiesmaier A.
[6] Spinsante S (2016), Gambi E (2016), Montanini L (2016), Raffaeli L (2016).
[7] Zeiger F (2015); Gazis V; Goertz; Huber; Leonardi; Mathioudakis; Wiesmaier.