ACADEMICIA

ISSN: 2249-7137 Vol. 11, Issue 10, October 2021 Impact Factor: SJIF 2021 = 7.492

ACADEMICIA
An International Multidisciplinary Research Journal
(Double Blind Refereed & Peer Reviewed Journal)

DOI: 10.5958/2249-7137.2021.02182.0

AN OVERVIEW ON ISSUES AND ENABLING TECHNOLOGIES IN IOT MIDDLEWARE

Ms Anuska Sharma*

*SOEIT, Sanskriti University,
Mathura, Uttar Pradesh, INDIA
Email id: anushka@sanskriti.edu.in

ABSTRACT

The Internet of Things (IoT) enables humans and computers to learn from and interact with billions of items such as sensors, actuators, services, and other Internet-connected gadgets. The implementation of IoT technologies will allow for seamless integration of the cyber and physical worlds, radically altering and empowering human interaction with the planet. Middleware, which is generally defined as a software system intended to be the intermediate between IoT devices and applications, is a crucial technology in the implementation of IoT systems. In this article, we first demonstrate the necessity for an IoT middleware by demonstrating an IoT application for real-time blood alcohol level prediction utilizing wristwatch sensor data. After that, a survey of the capabilities of current IoT middleware is conducted. We also undertake a comprehensive examination of the difficulties and enabling technologies in creating IoT middleware that embraces the heterogeneity of IoT devices while still supporting the key components of composition, flexibility, and security in an IoT system.

KEYWORDS: *Internet of Things, Middleware, Privacy, Service Discovery, Security.*

1. INTRODUCTION

The Internet of Things (IoT) is an area that, after the Internet, represents the next most exciting technological revolution. IoT will open up a world of possibilities and influence in every part of the globe. We can create smart cities with IoT, where parking, urban noise, traffic congestion, street lighting, irrigation, and trash can all be monitored in real time and controlled more efficiently. We can construct secure and energy-efficient smart houses. We can create smart ecosystems that monitor air and water pollution automatically and allow for early detection of earthquakes, forest fires, and other catastrophic catastrophes [1]. Manufacturing can be transformed by IoT, making it leaner and smarter. According to CBS News, over 600 bridges have failed in the United States since 1989. Every state has a significant number of bridges that pose a serious threat to motorists. Sensors enabled by the Internet of Things can monitor vibrations and material conditions in bridges (as well as buildings and historical sites) and give early warning, potentially saving many lives.

In almost every business sector conceivable, the SQ.Z. Shang works at the University of Adelaide's School of Computer Science, SA 5005, and Australia timely manage objects to create seamless integration of the physical and cyber worlds. There are many IoT middleware and connection protocols under development, and the number is growing by the day. Popular connection protocols developed especially for IoT devices include Message Oriented Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), and BLE (Bluetooth Low Energy) [2]. However, the variety of IoT connection protocols and middleware are making it difficult to connect IoT devices and understand the data they gather. The fact that each IoT middleware promotes a distinct programming abstraction and architecture for accessing and connecting to IoT devices adds to the confusion. The idea of virtual sensor, which is defined in XML and implemented with a matching wrapper, is given as the primary abstraction for creating and connecting a new IoT device in the Global Sensor Network (GSN) project, for example. The primary abstraction of the TerraSwarm project is an accessory design pattern implemented in Javascript. There is no high-level abstraction for encapsulating a new device type in the Google Fit project [3]. The system is pre-programmed to support a certain range of IoT devices that may be accessed through REST APIs. Extending Google Fit's FitnessSensorService class to accommodate an IoT device that isn't currently supported needs professional Java programming expertise. According to Zachariah et al. in their paper “The Internet of Things Has a Gateway Problem,” the current state-of-the-art support for IoT application development is application specific, which is equivalent to the scenario where each IoT device requires a different web browser to connect to the Internet. In this article, we examine the most cutting-edge middleware options for implementing IoT applications.are some of the surveys on IoT middleware that have been published. To our knowledge, these studies only look at IoT middleware from a few angles, and none of them address the more recent trend of light-weight plug-and-play or cloud-based IoT middleware.

The goal of this study is to get a better knowledge of current IoT middleware research and difficulties. The following are the paper's major contributions. The rest of this paper is laid out as follows. We first argue for the necessity for an IoT middleware based on our experience developing a real-time BAC predictor utilizing data from wristwatch sensors .We next go through our observations on the three main software designs for IoT middleware and provide a comparison and contrast of the three architectures [4] . We survey eight existing IoT middleware systems to see how well they fulfill the key functionalities required by BAC-like IoT applications, such as device abstraction for data collection, composition for visualization and analysis, service discovery for opportunistic integration, security and privacy. We compare these three kinds of IoT middleware by showing how to use GSN, Google Fit, and Ptolemy Accessor Host to gather data from a Phidgets sensor. The main research difficulties in creating an IoT middleware that allows a scientist or a health professional to configure/compose a BAC-like IoT program that is flexible, open, and secure are then discussed [5]. Figure 1 shows infrastructure for data collection and analysis.

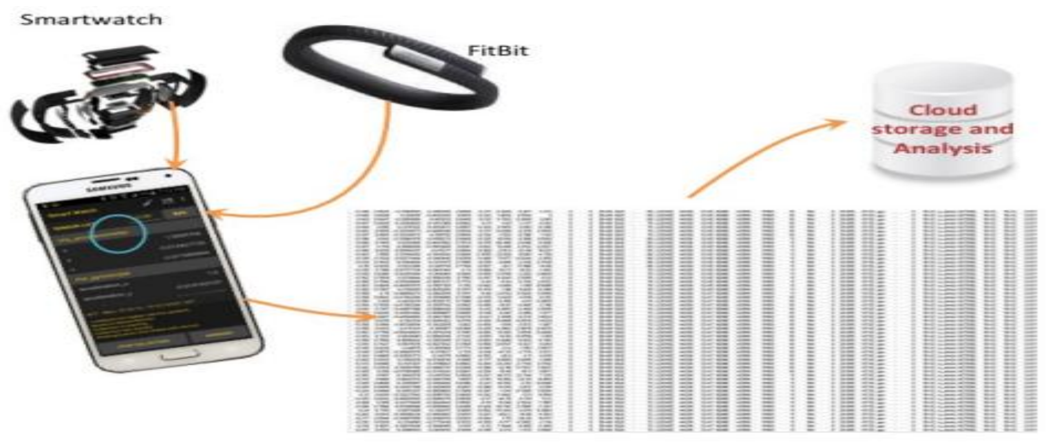

Figure 1: Infrastructure for Data Collection and Analysis.

Ambient data gathering and analytics, as well as real-time reactive applications, are two types of IoT applications. The first kind of application collects sensor data from a range of sensors (e.g., wearable devices), processes it offline to produce actionable information (e.g., a model), and then uses the model to forecast fresh data received from the sensor in the future. Real-time reactive systems, such as autonomous vehicles or industrial processes, fall under the second group of applications. These systems make real-time choices based on observed sensor readings. The first group of applications is quickly expanding, particularly in the healthcare sector, where customized health tracking and monitoring has become critical to providing better and more economical treatment.

Based on our experience developing an ambient data collecting and analytics IoT application that can predict Blood Alcohol Content (BAC) utilizing wristwatch sensor data , we justify the need for an open, lightweight, secure IoT middleware in this section. We'll go through the rationale for developing this IoT application, as well as how it, and all other IoT apps in this category, may benefit from an IoT middleware, in the sections below. Drunk driving is a serious issue that affects people all around the globe. This issue is a danger not just too intoxicated drivers, but also to pedestrians and other motorists. It may be difficult to assess one's own alcoholism at hazardous levels of drinking. It would be preferable to get a definite BAC measurement, or just a binary response: "drunk" or "not drunk". Infrastructure for Data Collection and Analysis is already available, however it is not discrete and requires the user to take intentional steps.

The second method is to manually calculate BAC using a Smartphone application, although this requires more effort from the user (remembering how many drinks they've had in a social environment). To be practical, some kind of non-invasive and accurate monitoring device that would alert users if they get too drunk would be beneficial. By connecting with the vehicle's ignition mechanism, this technology may also be used to alert friends and relatives, as well as prohibit the drinker's car from starting. We built a safe IoT application from the ground up to explore the prediction of intoxication lambent data gathering and analytics, as well as real-time reactive applications, are two types of IoT applications [6]. The first kind of application collects sensor data from a range of sensors (e.g., wearable devices), processes it offline to produce actionable information (e.g., a model), and then uses the model to forecast fresh data received from the sensor in the future. Real-time reactive systems, such as autonomous vehicles or industrial processes, fall under the second group of applications.

These systems make real-time choices based on observed sensor readings. The first group of applications is quickly expanding, particularly in the healthcare sector, where customized health tracking and monitoring has become critical to providing better and more economical treatment. Based on our experience developing an ambient data collecting and analytics IoT application that can predict Blood Alcohol Content (BAC) utilizing wristwatch sensor data , we justify the need for an open, lightweight, secure IoT middleware in this section. We'll go through the rationale for developing this IoT application, as well as how it, and all other IoT apps in this category, may benefit from an IoT middleware, in the sections below. Drunk driving is a serious issue that affects people all around the globe. This issue is a danger not just to intoxicated drivers, but also to pedestrians and other motorists. It may be difficult to assess one's own alcoholism at hazardous levels of drinking. It would be preferable to get a definite BAC measurement, or just a binary response: "drunk" or "not drunk", compact breathalyzers are probably the best choice. Infrastructure for Data Collection and Analysis is already available, however it is not discrete and requires the user to take intentional steps.

The second method is to manually calculate BAC using a Smartphone application, although this requires more effort from the user (remembering how many drinks they've had in a social environment). To be practical, some kind of non-invasive and accurate monitoring device that would alert users if they get too drunk would be beneficial. By connecting with the vehicle's ignition mechanism, this technology may also be used to alert friends and relatives, as well as prohibit the drinker's car from starting. We built a safe IoT application from the ground up to explore the prediction of intoxication level using wristwatch sensor data. The gathered data must be able to be kept locally as well as sent to a cloud storage system for analysis. It's critical to have local storage to prevent the unexpected latencies that come with wireless data transfer to the cloud. Data must be protected not just while it is in storage, both locally and in the cloud, but also while it is in transit. From the periphery to the cloud data center, end-to-end security is required. Following data collection, the data is analyzed in the cloud to see whether there is any correlation between the sensor readings and the recorded BAC levels. To do this, the data must first be pre-processed and displayed. The gathered sensor data is then subjected to a variety of machine learning algorithms in order to provide the best accurate prediction [7]

The basic infrastructure for a generic data gathering and processing system is shown in Figure 1. The data collection application on the Smartphone (also known as the gateway) uses a set of Java classes to handle the low-level details of the data collection process, such as managing the various threads for collecting sensor values from the Microsoft Band smart watch (also known as the edge device) or other devices like Fit bit. Before transferring the gathered data to the cloud for archiving, the data collecting program does some aggregation. The data analyses is done completely in the cloud, which is equipped with a high-performance computing engine as well as a variety of big data analytics and visualization tools. Once a model has been created using data analytics, it is stored and used as a predictor in the BAC application.evel using wristwatch sensor data. The gathered data must be able to be kept locally as well as sent to a cloud storage system for analysis. It's critical to have local storage to prevent the unexpected latencies that come with wireless data transfer to the cloud. Data must be protected not just while it is in storage, both locally and in the cloud, but also while it is in transit. From the periphery to the cloud data center, end-to-end security is required.

Following data collection, the data is analyzed in the cloud to see whether there is any correlation between the sensor readings and the recorded BAC levels. To do this, the data must first be pre-processed and displayed. The gathered sensor data is then subjected to a variety of machine learning algorithms in order to provide the best accurate prediction. The basic infrastructure for a generic data gathering and processing system is shown The data collection application on the Smartphone (also known as the gateway) uses a set of Java classes to handle the low-level details of the data collection process, such as managing the various threads for collecting sensor values from the Microsoft Band smart watch (also known as the edge device) or other devices like Fit bit. Before transferring the gathered data to the cloud for archiving, the data collecting program does some aggregation. The data analyses are done completely in the cloud, which is equipped with a high-performance computing engine as well as a variety of big data analytics and visualization tools. Once a model has been created using data analytics, it is stored and used as a predictor in the BAC application.

2. DISCUSSION

In order to connect and receive data from all possible sensors on the Microsoft Band wristwatch, it must first be virtualized as a software component to the BAC Smartphone application. In order to shield users from the low-level implementation details of networking protocols and communication capabilities of various physical sensors, a device abstraction component is required. The BAC application and the physical device must be able to communicate in real time. Data is often supplied from devices in endless streams in time-stamped order. As a result, a stream, event processing, or aggregation service is a critical component. Stream processing detects complicated events and converts the gathered data (which is typically in huge quantities) into usable information. Aggregation may help you get more useful data for your research. When gathering accelerometer data, for example, the three most recent values were averaged using linear weighting rather than simply taking the most recent value.

Users will require a monitoring or visualization service to monitor/control the status of the physical devices, as well as to regulate when and how frequently the gathered data should be archived to the cloud for future analysis or processing. This component should also offer notification and subscription services to customers in order to provide IoT status, in this instance an alarm for being drunk, on a timely basis. The security and privacy component is required to ensure the integrity of the collected data (stream) and to ensure that the user's privacy is not violated. An IoT application can generate large amounts of data that must be processed and archived, so a ubiquitous connection to a cloud infrastructure is required for data analytic and archiving. Users should be able to save gathered data to their preferred storage media and should only be able to connect to authenticated/certified IoT devices. A composition engine (also known as a rule engine in certain systems) is required to allow users to mix analytics services from the cloud, data services from other gateways (PhidgetInterfaceKit, Adruino), or other IoT devices (car's ignition device) without having to do any low-level programming data collecting and analytics system for monitoring environmental pollution in a building would use a similar set of computing units as the BAC prediction application, with the exception that the edge sensors would be Mica mote, and the gateway will be a desktop or laptop.

The information gathered will be sent to cloud storage or a backend database. For analytics, a comparable collection of analytical and visualization tools is required. In conclusion, the logical requirements for both environment and BAC monitoring are the identical. Having to build two distinct apps for each of the aforementioned applications with dedicated resources not only increases development costs and time, but also hinders the creation of new IoT applications for safe and privacy-preserving data and IoT device sharing.IoT Middleware on the Cloud for a Variety of Applications. As a result, an open, lightweight, flexible, and secure IoT middleware that acts as a bridge between various IoT devices and applications is required. Without any low-level programming, a scientist or a health professional may configure/compose a new secure IoT application for conducting data gathering and analysis appropriate to his or her context.

2.1. Application:

By creating a pluggable actor or downloading it from a common repository, you may create IoT middleware. Both service and actor-based IoT middleware designs do not impose a specific standard for interoperability across IoT devices, such as Restful API or BLE. By supporting a specific programming paradigm or device abstraction, they both welcome the variety of IoT devices. Interoperability in cloud-based architecture, on the other hand, is accomplished via the adoption of particular standards. When a cloud provider's service is terminated, cloud-based middleware may cease to function entirely. A good example of this is Google Nest .While all three designs enable security and privacy to some extent, cloud-based architecture necessitates users' confidence in the cloud provider to protect their data's privacy and integrity. Users aren't offered any other options beyond those recommended by the cloud. Users in service and actor-based architectures have control over how and where data is kept. Because the middleware cannot be integrated inside the physical device and data transferred between physical devices and the middleware may be hacked, there is a weak security connection between physical devices and the middleware in both service and cloud-based architectures. IoT applications are often used in a changing and unpredictable environment. IoT devices, for example, may run out of battery power and cease to function, and connection between devices and gateways may be severed at any moment. The middleware must have a service discovery component so that new services can be made accessible on demand and failing services may be dynamically replaced to ensure a particular level of service quality (QoS). If the present gateway is likely to lose connectivity, the physical devices may connect to a new gateway of comparable quality. Currently, only service-based middleware provides a restricted version of service discovery.

2.2. Advantage:

Web-based service Wrapping IoT devices as Web services with the SDK tool kit and training may restrict the kinds of IoT devices that can be deployed and controlled in this platform, since Web service is a heavy protocol to operate on energy and capabilities limited IoT devices. For processing and preservation, all gathered data is sent to the Hydra middleware. On IoT devices, no local processing or aggregation of acquired data is available. This is problematic for certain BAC-like applications that need real-time analysis of acquired data to identify crucial events (e.g., an old person's fall). Hydra IoT applications must be built by a programmer, and it is not a platform that allows users to easily find, build, and deploy BAC-style data gathering and analysis applications. Hydra is therefore more suited to enterprise-level IoT applications that establish a long-term and tight connection with a fixed set of IoT devices that the platform currently supports. The virtual sensor abstraction is the core idea, which allows users/developers to declaratively define XML-based deployment descriptors for deploying a sensor. This is comparable to the deployment descriptors idea used in the J2EE server to deploy enterprise beans.

The GSN design is similar to that of J2EE, in that each container may host many virtual sensors, and the container offers functionality for sensor lifecycle management, such as persistency, security, notification, resource pooling, and event processing. One or more data streams are sent into the virtual sensor, which are then processed according to the XML standard. The sampling rate of the data, the kind and location of the data stream, the data's persistency, the data's output format, and the SQL processing logic for the data stream are all factors to consider. A wrapper is assigned to each input stream. When the physical sensor is initially started, the wrapper software defines I the network protocol to use to connect, interact, and communicate with it, ii) what to do in order to read data from the sensor, and iii) what to do with the data after it is received from the sensor. If the virtual sensor's permanent storage property is set to "true" in the XML specification, GSN offers a SQL-based database that saves all raw sensor data. Furthermore, each virtual sensor has a key-value pair that may be found and registered in GSN.

The flexibility to build a platform-specific wrapper allows the system to work with a variety of sensors. To add a new kind of sensor to the platform, the user must first understand how to create an XML descriptor for the physical sensor and, if one is not already available, offer a Java wrapper implementation. To show the capabilities offered by GSN's device abstraction, we demonstrate the construction of Phidgets sensors in the next paragraph. Because we had prototype implementations of Phidgets sensors in all three kinds of middleware that we saw, we choose to present Phidgets sensors implementation for the remainder of the article. The light and sound sensor data we gathered in Phidgets are similar to sensor data acquired from a smart watch in terms of properties.

Adding a Phidgets sensor (IoT device) as a new virtual sensor in GSN requires the development of a deployment file (see Figure 6) and a wrapper class that can operate as a thread and consume stream data according to the settings provided in the XML deployment file. The storage media for the gathered data is specified by the virtual-sensor-name tag in the deployment file. The processing-class tag defines the virtual sensor's Java class, which in this instance is Phidget Virtual Sensor. The output-structure tag defines the data collection's structure. It's the music and the light in this instance, and they're both of the double kind. The stream tag defines how the program must enable real-time interaction between the physical device and the application. The sampling rate and the processing logic on the gathered data, for example, are defined using the attributes sampling-rate and query tag.

While GSN offers scalable servers for sensor data collecting and storage, it does not provide tools for composing or interpreting the data beyond displaying it on a Web application provided by GSN. It also doesn't allow multi-vendor device composition through the XML descriptor. The expanded GSN, which is part of the OpenIoT project, does, however, provide a limited composition capability. When data from different IoT devices has to be gathered and merged, a programmer must build a domain specific application in GSN. External applications may use Restful or Web service APIs to access virtual sensors stored on GSN. There is support for a rudimWeb-based service Wrapping IoT devices as Web services with the SDK tool kit and training may restrict the kinds of IoT devices that can be deployed and controlled in this platform, since Web service is a heavy protocol to operate on energy and capabilities limited IoT devices.

For processing and preservation, all gathered data is sent to the Hydra middleware. On IoT devices, no local processing or aggregation of acquired data is available. This is problematic for certain BAC-like applications that need real-time analysis of acquired data to identify crucial events (e.g., an old person's fall). Hydra IoT applications must be built by a programmer, and it is not a platform that allows users to easily find, build, and deploy BAC-style data gathering and analysis applications. Hydra is therefore more suited to enterprise-level IoT applications that establish a long-term and tight connection with a fixed set of IoT devices that the platform currently supports. The virtual sensor abstraction is the core idea, which allows users/developers to declaratively define XML-based deployment descriptors for deploying a sensor. This is comparable to the deployment descriptors idea used in the J2EE server to deploy enterprise beans. The GSN design is similar to that of J2EE, in that each container may host many virtual sensors, and the container offers functionality for sensor lifecycle management, such as persistency, security, notification, resource pooling, and event processing.

One or more data streams are sent into the virtual sensor, which are then processed according to the XML standard. The sampling rate of the data, the kind and location of the data stream, the data's persistency, the data's output format, and the SQL processing logic for the data stream are all factors to consider. A wrapper is assigned to each input stream. When the physical sensor is initially started, the wrapper software defines I the network protocol to use to connect, interact, and communicate with it, ii) what to do in order to read data from the sensor, and iii) what to do with the data after it is received from the sensor. If the virtual sensor's permanent storage property is set to "true" in the XML specification, GSN offers a SQL-based database that saves all raw sensor data. Furthermore, each virtual sensor has a key-value pair that may be found and registered in GSN.

The flexibility to build a platform-specific wrapper allows the system to work with a variety of sensors. To add a new kind of sensor to the platform, the user must first understand how to create an XML descriptor for the physical sensor and, if one is not already available, offer a Java wrapper implementation. To show the capabilities offered by GSN's device abstraction, we demonstrate the construction of Phidgets sensors in the next paragraph. Because we had prototype implementations of Phidgets sensors in all three kinds of middleware that we saw, we choose to present Phidgets sensors implementation for the remainder of the article[8]. The light and sound sensor data we gathered in Phidgets are similar to sensor data acquired from a smart watch in terms of properties.

Adding a Phidgets sensor (IoT device) as a new virtual sensor in GSN requires the development of a deployment file and a wrapper class that can operate as a thread and consume stream data according to the settings provided in the XML deployment file. The storage media for the gathered data is specified by the virtual-sensor-name tag in the deployment file. The processing-class tag defines the virtual sensor's Java class, which in this instance is Phidget Virtual Sensor. The output-structure tag defines the data collection's structure. It's the music and the light in this instance, and they're both of the double kind. The stream tag defines how the program must enable real-time interaction between the physical device and the application. The sampling rate and the processing logic on the gathered data, for example, are defined using the attributes sampling-rate and query tag., which extends GSN's AbstractWrapper class and uses this XML descriptor as input [9].

While GSN offers scalable servers for sensor data collecting and storage, it does not provide tools for composing or interpreting the data beyond displaying it on a Web application provided by GSN. It also doesn't allow multi-vendor device composition through the XML descriptor. The expanded GSN, which is part of the OpenIoT project, does, however, provide a limited composition capability. When data from different IoT devices has to be gathered and merged, a programmer must build a domain specific application in GSN. External applications may use Restful or Web service APIs to access virtual sensors stored on GSN. There is support for a rudimentary kind of service discovery based on dictionary lookup. A login account protects user information. All captured data is sent to the middleware for processing and archiving, much as Hydra. GSN isn't intended to run on low-power, low-processing-power IoT gateways like Smartphone or Raspberry Pi, thus no local data processing or aggregation is done. Declarative sensor capability definition through XML descriptor file is a step in the right direction for fast development of BAC-like applications via automated wrapper class generation from the descriptor file [10].

It's a cloud-based IoT middleware that allows customers to manage their fitness data and create fitness applications all from one place. It aims to achieve the same objective as Apple's Health Kit. The system includes a fitness store, which is a cloud storage service that saves data from a number of devices and applications (similar to Firebase, a JSON-based document server). A sensor framework is a collection of APIs that allow third-party IoT devices to connect to its store. It offers APIs for subscribing to a certain fitness data type or source (e.g., Fitbit or Smartwatch), as well as APIs for accessing past data and permanent recording of the information. entary kind of service discovery based on dictionary lookup. A login account protects user information. All captured data is sent to the middleware for processing and archiving, much as Hydra. GSN isn't intended to run on low-power, low-processing-power IoT gateways like Smartphone or Raspberry Pi, thus no local data processing or aggregation is done. Declarative sensor capability definition through XML descriptor file is a step in the right direction for fast development of BAC-like applications via automated wrapper class generation from the descriptor file. It’s a cloud-based IoT middleware that allows customers to manage their fitness data and create fitness applications all from one place. It aims to achieve the same objective as Apple's HealthKit. The system includes a fitness store, which is a cloud storage service that saves data from a number of devices and applications (similar to Firebase, a JSON-based document server). A sensor framework is a collection of APIs that allow third-party IoT devices to connect to its store. It offers APIs for subscribing to a certain fitness data type or source (e.g., Fit bit or Smartwatch), as well as APIs for accessing past data and permanent recording of the information [10].

2.3.Working:

service on the web Wrapping IoT devices as Web services with the SDK tool kit and training may restrict the kinds of IoT devices that can be deployed and controlled under this platform, since Web services are a heavy protocol to operate on power and capabilities limited IoT devices. To be processed and archived, all gathered data is sent to the Hydra middleware. On IoT devices, there is no way to analyze or aggregate the data gathered locally. This is problematic for certain BAC-like applications that need real-time analysis of gathered data to identify crucial events (such as an old person's fall). The Hydra IoT application must be handmade by a programmer, and it is not a platform that allows users to easily find, build, and deploy BAC-like data collecting and analysis applications. Hydra is therefore more suited to enterprise-level IoT applications that have a long-term and tight connection with a fixed set of IoT devices that the platform currently supports. GSN, is a service-based Internet of Things that seeks to offer a consistent platform for flexible integration, sharing, and deployment of heterogeneous IoT devices.

The virtual sensor abstraction is the core idea, which allows users/developers to define XML-based deployment descriptors declaratively to deploy a sensor. This is analogous to the notion of deployment descriptors in J2EE server, which are used to deploy enterprise beans. GSN's design is similar to that of J2EE in that each container may host many virtual sensors and the container offers functionality for sensor lifecycle management, such as persistency, security, notification, resource pooling, and event processing. One or more data streams are sent into the virtual sensor, which are then processed according to the XML standard. The data sampling rate, the kind and location of the data stream, the data's persistency, the data's output format, and the SQL processing logic for the data stream are all factors to consider. Wrappers are assigned to each input stream. When the physical sensor is initially started, the wrapper software defines (i) the network protocol to use to connect, interact, and communicate with it, ii) what to do to read data from the sensor, and iii) what to do with the data after it has been received from the sensor.

If the permanent storage property of the virtual sensor is set as "yes" in the XML specification, GSN offers a SQL-based database that saves all raw sensor data. Each virtual sensor also has a key-value pair that may be found and registered in GSNThe system can connect with sensors of various kinds thanks to the flexibility to install a platform-specific wrapper. To add a new kind of sensor to the platform, the user must first understand how to create an XML descriptor for the physical sensor, as well as offer a Java wrapper implementation if one is not already available. To showcase the capabilities offered by GSN's device abstraction, we'll show how to create Phidgets sensors in the next paragraph. Because we had prototype implementations of Phidgets sensors in all three kinds of middleware that we saw, we chose to present them for the remainder of the article.2016 IEEE, 2327-4662 Personal use is allowed, but reprinting or dissemination needs IEEE approval. The light and sound sensor data we gathered in Phidgets are similar to sensor data acquired from a smart watch in terms of features. Adding a Phidgets sensor (IoT device) as a new virtual sensor in GSN necessitates the development of a deployment file, as illustrated as well as the construction of a wrapper class that can operate as a thread and consume stream data according to the settings provided in the XML deployment file. The storage media for the gathered data is defined by the virtual-sensor-name tag in the deployment file. The processing-class tag defines the virtual sensor's Java class, which is Phidget VirtualSensor in this instance. The output-structure tag describes the data collection's structure. It's the music and light in this instance, and they're both of the double kind. The stream tag describes how the physical device and the program must communicate in real time.

The sample rate and processing logic on the gathered data, for example, are defined using the sampling-rate property and the query tag. Figure 7 shows a portion of the wrapper class, which extends GSN's Abstract Wrapper class and takes input from this XML descriptor. While GSN offers scalable servers for sensor data collecting and storage, it does not provide tools for composing or interpreting the data beyond displaying it on a Web application. It also doesn't allow for the XML descriptor to be used to compose multi-vendor devices. The Open IoT project's expanded GSN does, however, provide limited composition capabilities. When data has to be gathered and merged from a variety of IoT devices, a programmer must build a domain specific application in GSN. Restful or Web service APIs allow other applications to connect to virtual sensors stored on GSN. Service discovery based on dictionary lookup is supported to a limited extent.

A login account keeps user information safe. All collected data is sent to the middleware, which processes and archives it, much as Hydra. Because GSN is not intended for use in IoT gateways with limited power and processing capabilities, such as Smartphone or Raspberry Pi, no local data processing or aggregation is done. The automated development of the wrapper class from the descriptor file is a step in the right direction for fast building of BAC-like applications. IoT Middleware on the Cloud Google Fit is an open Internet of Things ecosystem. It’s a cloud-based IoT middleware that allows customers to manage their fitness data and develop fitness applications all from one place. Apple's Health Kit has a similar objective. A fitness store is included in the system, which is a cloud storage service (similar to Firebase, a JSON-based document server) that saves data from various devices and applications. A sensor framework is a collection of APIs that allow third-party IoT devices to connect to a company's shop. APIs for subscribing to a certain fitness data type or source (e.g., Fit bit or Smartwatch), APIs for accessing previous data, and APIs for permanent recording of the s are just a few examples.

3. CONCLUSION

The World Wide Web has gone through many transformations, from traditional linking and sharing of computers and documents, to a platform for conducting businesses and connecting people via social media, and now the emerging paradigm of connecting billions of physical objects (Internet of Things) to empower human interaction with both the physical and virtual worlds in an unprecedented way. In this survey paper, we have analyzed three key IoT middleware architectures ranging from consumer centric cloud-based architectures, light-weight actor-based architectures, and heavy weight service-based architectures. We outlined four key challenges in developing an IoT middleware which are: 1) a light-weight middleware platform that can provide similar services when deployed on power constrained IoT devices as well as in desktop computers and cloud infrastructure; 2) a composition engine that is intuitive and not application specific; 3) a security mechanism that can operate in a resource constrained environment and yet can achieve similar guarantee as Internet security; and 4) a semantic-based IoT device/service discovery that goes beyond discovery of domain names and IP addresses. We elaborate on two non-ontological solutions for addressing key challenges in IoT service discovery. The first approach is adapted from existing works in Web service search engines and the second approach is based on machine learning and recommendation techniques. Finally, in the IoT security domain, we believe emerging techniques such as privacy by design, differential privacy, and light weight public key cryptography will form the building blocks for security in IoT middleware.

REFERANCES:

1. M. A. Khan and K. Salah, “IoT security: Review, blockchain solutions, and open challenges,” *Futur. Gener. Comput. Syst.*, 2018.
2. R. S. Sinha, Y. Wei, and S. H. Hwang, “A survey on LPWA technology: LoRa and NB-IoT,” *ICT Express*. 2017.
3. A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, “Blockchain and iot integration: A systematic survey,” *Sensors (Switzerland)*. 2018.
4. O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. N. Hindia, “An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges,” *IEEE Internet Things J.*, 2018.
5. A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, “On blockchain and its integration with IoT. Challenges and opportunities,” *Futur. Gener. Comput. Syst.*, 2018.
6. M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, “Deep learning for IoT big data and streaming analytics: A survey,” *IEEE Communications Surveys and Tutorials*. 2018.
7. M. Frustaci, P. Pace, G. Aloi, and G. Fortino, “Evaluating critical security issues of the IoT world: Present and future challenges,” *IEEE Internet Things J.*, 2018.
8. A. Oussous, F. Z. Benjelloun, A. Ait Lahcen, and S. Belfkih, “Big Data technologies: A survey,” *Journal of King Saud University - Computer and Information Sciences*. 2018.
9. J. Martín-Gutiérrez, C. E. Mora, B. Añorbe-Díaz, and A. González-Marrero, “Virtual technologies trends in education,” *Eurasia J. Math. Sci. Technol. Educ.*, 2017.
10. J. Rybicka, A. Tiwari, and G. A. Leeke, “Technology readiness level assessment of composites recycling technologies,” *J. Clean. Prod.*, 2016.